Universidad de La Laguna

Grado en Ingeniería Electrónica, Industrial y Automática

TRABAJO FIN DE GRADO

Título: Desarrollo de un sistema controlador para una red domótica inalámbrica

Autor: Javier Pérez Martín

Tutor: Alberto Hamilton Castro

AUTORIZACIÓN

D. Alberto Hamilton Castro, con N.I.F. 43773884P, profesor Titular de Universidad adscrito al Departamento de Ingeniería de Sistemas y Automática de la Universidad de La Laguna, como tutor **certifica** que la presente memoria titulada: "Desarrollo de un sistema controlador para una red domótica inalámbrica" ha sido realizada bajo su dirección por D. Javier Pérez Martín, con N.I.F. 78642574P.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 4 de marzo de 2016.

ÍNDICE

1. INTRODUCCIÓN

1.1.Objeto del proyecto

El objeto del proyecto será la configuración y el conexionado de una pequeña red domótica basada en el protocolo Zigbee. Los dispositivos utilizados para este propósito son los populares Xbee, de la compañía Digi International. Un ordenador pondrá en funcionamiento la red desde un sistema GNU/Linux que realizará las labores control del sistema montado. Para ello, hará uso de un programa demonio (con funcionamiento continuo) que permita al usuario poner en marcha el sistema o realizar distintas configuraciones de testeo para el correcto funcionamiento de la red. Ésta, en nuestro caso particular, se ha montado para implementar un sistema de alarma que sea portátil, inalámbrico y de bajo consumo para poder ubicarlo en cualquier lugar en un radio de 30 metros.

1.2.Abstract

The objetive of this proyect will be the configuration and the electrical conexion of a small home automation network based on the Zigbee protocol. The devices used in this proyect are the well-known Xbee, from the Digi International Company. If we want to make this network works, we need a computer with GNU/Linux for control the system. The network control will use a demon software (with continuous operation) whose mission is run the system or test it. In our particular case, this network has been mounted for create an alarm system that will be portable, wireless and energy-saving. In this way, we can place it anywhere around 30 metres.

1.3.Introducción

Digi International fue fundada en el año 1985 (con el nombre DigiBoard) y en la actualidad se dedica a comercializar productos relacionados con la comunicación entre dispositivos. Entre estos destacamos los Xbee, mecanismos capaces de comunicar mediante radiofrecuencia a circuitos separados una cierta distancia entre sí. Esto presenta grandes ventajas a la hora de comunicar diferentes circuitos electrónicos distantes entre sí, como la reducción del cableado utilizado o la posibilidad de trasladar un circuito de un lugar a otro sin necesidad de cablear nuevamente. En el mundo de la domótica no depender de cables a la hora de administrar o controlar una red de sensores y actuadores es muy conveniente, ya que seremos capaces de automatizar una vivienda aportándole servicios de bienestar, de comunicación o de seguridad (como el que nos ocupa) de una manera sencilla, eficiente y práctica. La integración domótica en nuestro hogar, ya sea cableada o inalámbrica, nos abre posibilidades inexistentes hasta hace

unos años. Ser capaces de controlar, por ejemplo, el encendido o el apagado de las distintas luminarias, la apertura o el cierre de las persianas o la medir la cantidad de luz que exista en una habitación (mediante sensores fotoeléctricos o luxómetros) nos ayudará a optimizar energéticamente nuestra vivienda de manera totalmente automatizada. Además, este ejemplo de domótica también nos sirve para explicar como un entorno automatizado puede ayudar a corregir pequeños errores humanos. La vivienda puede ser programada para apagar las luces a una hora establecida o pasado cierto tiempo después de determinada acción. Nuestro sistema de control podría ser capaz de detectar que el usuario al salir de la vivienda ha dejado luces encendidas y actuar en consecuencia. Para lo primero podría basarse en el periodo de tiempo transcurrido entre la apertura y el cierre de la puerta principal. Asimismo, los avances en telefonía móvil permiten abrir nuevas posibilidades para el mundo de la domótica, pudiendo controlar a través de un smartphone los distintos actuadores que tengamos distribuidos en nuestro hogar. En este sentido el usuario podría ver, a través de una aplicación o mediante un correo electrónico que luces ha dejado encendidas y, además, podría actuar en consecuencia enviando órdenes a su sistema de control desde su dispositivo móvil.

No obstante, desarrollos como el anterior empiezan a ser cada vez más habituales en el mundo de la domótica y en nuestro caso queríamos hacer algo distinto, plantear el proyecto con un circuito sencillo y a su vez práctico. En este sentido se pretende que podamos intercambiar los sensores y actuadores por otros a nuestra conveniencia para recrear otros circuitos. Por ejemplo, se podría cambiar el sentido de seguridad de nuestro proyecto por un sentido más práctico cambiando simplemente de actuador. Sustituyendo el buzzer por un relé bien situado podemos encender o apagar cualquier cosa que se nos ocurra con solo entrar a una habitación o sencillamente acercando la mano a nuestro sensor.

2. CONOCIMIENTOS PREVIOS

En este segundo capítulo indagaremos en distintos aspectos del protocolo ZigBee y los dispositivos basados en este, los Xbee de Digi International. Comentaremos las topologías aceptadas y los diferentes modos de trabajo que nos permiten estos módulos, centrándonos en el usado durante la implementación, el modo API. Igualmente se hablará de la configuración previa que deberemos realizar a nuestro sistema GNU/Linux y de manera introductoria del resto de dispositivos utilizados: Arduino y Raspberry Pi.

2.1.Protocolo Zigbee

2.1.1. ¿Qué es Zigbee?

Creado por la organización sin ánimo de lucro **Zigbee Alliance**, el protocolo de comunicaciones Zigbee está basado en el estándar de comunicaciones para redes inalámbricas de área personal (WPAN) **IEEE 802.15.4**. Su utilidad radica en la facilidad para comunicar inalámbricamente, de forma segura y con baja tasa de envío de datos, dispositivos electrónicos de bajo consumo. Por ello, la domótica sobresale como el principal ambiente en el que destinar esta tecnología. Características como su fácil integración electrónica, el bajo consumo de los dispositivos y su topología en malla facilitan este uso.

Las comunicaciones se realizan en la banda de 2.4 GHz y, a diferencia de bluetooth, este protocolo no utiliza el FHSS. En lugar de eso, las comunicaciones se realizan a través de un único canal disminuyendo así considerablemente el ancho de banda ocupado. La velocidad de transmisión y la cantidad de dispositivos que puede contener una red Zigbee es de 256 Kb/s y 65535 equipos respectivamente. El alcance normal (en visión directa) de la antena dipolo es de 30 metros en interiores y 100 metros en el exterior, resultando muy práctico para realizar proyectos inalámbricos de corto alcance.

Una red Zigbee está compuesta por tres tipos de elementos: un coordinador, uno o varios routers y uno o varios dispositivos finales. Con ellos se puede desde reemplazar un cable por una comunicación serial inalámbrica, ahorrándonos el esfuerzo de cablear hasta lugares complicados o incluso permitiendo reubicar sensores o actuadores sin la mayor complicación.

- **Coordinador:** Es el que tiene la función más importante, crear la red y permitir a los nodos que funcionan como router o end device unirse a ella. Además, es el responsable de crear el canal de comunicaciones y establecer el identificador de red (PAN ID).

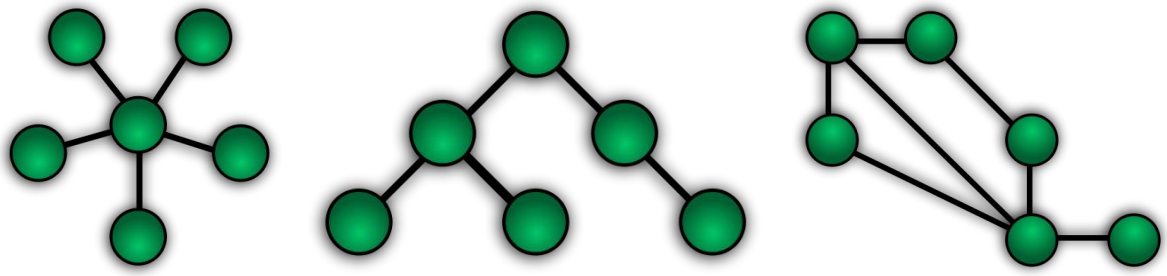
También es posible que los módulos que operen como coordinador asuman funciones de router y participen en el reparto de rutas a los paquetes.

- **Routers:** Se encargan de comunicarse con todos los dispositivos existentes en la red y participan activamente en el reparto de rutas. Además, determinan cuál de estas es la mejor para distribuir los mensajes.

- **End Devices:** Son aquellos que no pueden comunicarse con otros dispositivos finales. Solo podrán interactuar a través de su nodo padre (este puede ser un coordinador o un router). Además, no tienen la capacidad necesaria para participar en el reparto de rutas de distribución de paquetes. [2][7][16]

2.1.2. Topologías de red

Es a través de las direcciones que los dispositivos Xbee son capaces de comunicarse entre sí, por lo tanto, vienen ya de fábrica con una dirección de 64 bits, única e intransferible similar a las direcciones MAC de 48 bits de los dispositivos Ethernet. No obstante, para los algoritmos de ruteo se pueden utilizar direcciones de 16 bits asignadas por el Zigbee Coordinador. Estas son únicas en toda la red y permiten abarcar una menor cantidad de nodos que las anteriores. Gracias a esta asignación de direcciones se pueden establecer distintas rutas entre uno o varios nodos e interconectarlos de diferentes formas. El protocolo Zigbee reconoce varias maneras de interconectar los nodos existentes, siendo capaz de trabajar con las siguientes topologías de red: estrella, árbol y malla.



Estrella **Árbol** **Malla**

[Fig. 1] Topologías de red

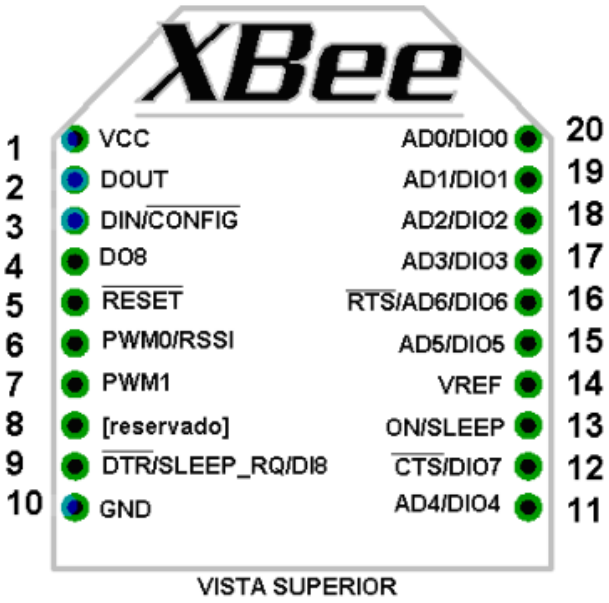
- Topología en estrella: El coordinador se situará en el centro de la red y el resto de dispositivos estarán conectados directamente a él, recibiendo así todas las comunicaciones que se realicen. El problema que surge con esta manera de operar es que un fallo en el punto central de la red dejará inoperativa la misma.

- Topología en árbol: Funciona como un conjunto de redes en estrella interconectadas entre sí. Esto es, por ejemplo, que el coordinador se encuentra conectado a varios routers y estos a su vez a uno o varios dispositivos finales. La principal desventaja aparece cuando el coordinador o alguno de los routers falla, ya que dejará inoperativa toda o parte de la red.

- Topología en malla: Es la más interesante para utilizar con Zigbee porque al tratarse de redes inalámbricas supera una de las principales desventajas de esta topología, el excesivo cableado. Cada nodo se unirá a varios, proporcionando así una mayor fiabilidad. Si uno de los dispositivos del camino falla siempre se podrá continuar con la comunicación haciendo que el Zigbee Coordinador establezca una nueva ruta y subsane el error.

2.1.3. Características y patillaje de Xbee

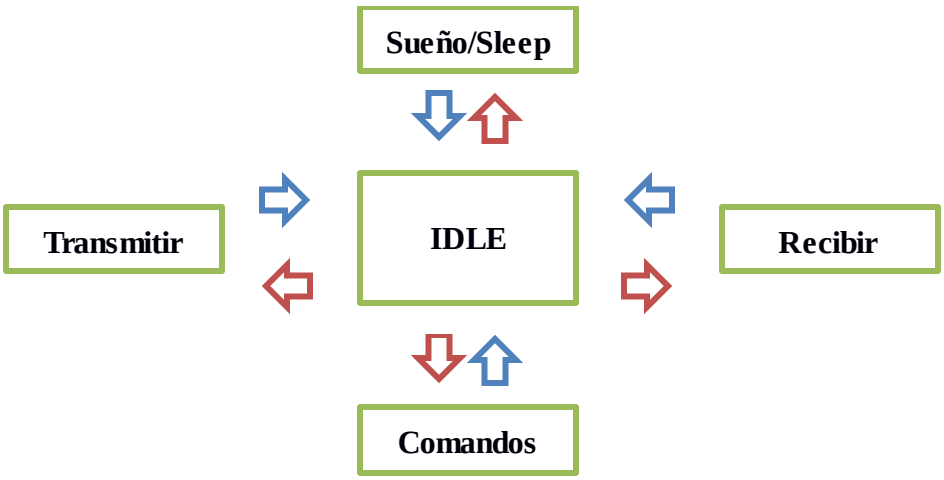
Existen varios módulos basados en el protocolo Zigbee desarrollados por la compañía Digi International. Los más destacados son los *Xbee Serie 1* por su gran capacidad de alcance y los *Xbee Serie 2* que permiten hacer redes más complejas, las llamadas MESH. Estas tienen una topología malla y permiten acceder a un punto remoto a través de los módulos intermedios. Por otro lado, uno de estos dispositivos dispone de 12 pines con opción de funcionar como entradas o salidas digitales o entradas analógicas. Se alimentan a una tensión de 3.3 voltios y disponen de pines de transmisión y recepción, los conocidos como UART. Su patillaje o pinout sería el siguiente: [11]



[Fig. 2] Diagrama de pines de un módulo Xbee

2.1.4. Modos de operación

Los Xbee tienen cinco modos de operación: idle, sueño, recibir, transmitir y comandos.



[3] Modos de operación

Modo Recibir/Transmitir: El módulo se encuentra en alguno de estos modos cuando le llega un paquete de radiofrecuencia a través de su antena o cuando envía información serial al buffer del pin número 3 (Data IN) para luego transmitirla por el pin 2 (Data Out). Esta transmisión de información puede ser directa o indirecta. En el primer caso el mensaje es enviado directamente a la dirección de destino. En el segundo caso la información se retiene durante cierto periodo de tiempo y solo se envía cuando es solicitada por la dirección de destino. Asimismo, y siguiendo con el modo de transmisión, es posible enviar mensajes mediante dos formas: unicast y broadcast.

- Unicast: Comunicación punto a punto. Quien recibe el mensaje debe enviar un paquete ACK de confirmación a la dirección de origen. Si esta última no recibiera la confirmación enviará el paquete tres veces más o hasta que reciba el ACK.
- Broadcast: Comunicación multipunto, entre un nodo y el resto. La diferencia con el modo de comunicación anterior es que en este caso no existe la confirmación mediante el paquete ACK.

Modo Sueño/Sleep: El módulo entrará en un modo de bajo consumo de energía cuando no se encuentre en uso. Para ello se deberán cumplir dos condiciones: sobrepasar el tiempo prefijado de reposo y se mantener activo el pin número 9 (Sleep_RQ). Asimismo, se pueden configurar varios modos de funcionamiento de los ciclos de sueño, encontrándose estos deshabilitados (estado reposo/recepción) por defecto y/o fijando a 0 el comando SM. Existen dos formas de activar los modos de sueño, mediante pin o mediante el envío de comandos.

- Pin de hibernación: Se habilita poniendo en estado lógico alto el pin número 9, Sleep_RQ. Mientras esté en este modo el consumo de energía será mínimo y el módulo finalizará cualquier transmisión, recepción o procedimientos de asociación que se encuentre realizando. La única manera de salir del modo hibernación es poniendo en estado lógico bajo el anteriormente nombrado pin número 9, ya que el dispositivo no responderá a ningún comando entrante.

- Pin Doze: Funciona de la misma manera que el modo hibernación con la peculiaridad de presentar un mayor consumo de energía y un menor tiempo de activación. La lógica para habilitar y deshabilitar este modo es similar a la del modo pin de hibernación. Si queremos despertar el módulo y que vuelva a transmitir o recibir información debemos poner en estado lógico bajo el Sleep_RQ (pin 9) y el CTS (pin 12).

- Cíclico remoto: Se habilita fijando el parámetro, o comando, SM a 4 y permite que el módulo revise periódicamente (mediante la interfaz de radiofrecuencia) si existe algún mensaje para él. Esto se refiere a que al finalizar cada ciclo de sueño el módulo se despertará y enviará al nodo padre de la red una solicitud para que este le transmita los datos que tenga para él en su buffer de salida. Esta solicitud se enviará a intervalos de tiempo determinados por el parámetro ST, periodo de sueño. Si el nodo padre no tiene datos para ser enviados al módulo que realiza la solicitud no transmitirá nada y por tanto,

el módulo regresará a su estado de sueño. En cambio, los transmitirá hasta que el tiempo ST se cumpla si existen datos para ser enviados.

- Cíclico remoto con pin para despertar: Su funcionalidad es la misma que en el modo anterior pero con la peculiaridad de que se puede forzar el despertar del módulo remoto. Esto se consigue mediante la interfaz de radiofrecuencia o poniendo en estado lógico bajo el pin número 9, Sleep_RQ. Al igual que en los casos anteriores, la duración que hallamos definido previamente para ST y que no ocurran actividades durante ese periodo nos marcará cuando el módulo vuelve al estado de sueño.

- Nodo padre: Sirve para configurar el módulo como tal. En este modo el nodo aceptará y mantendrá, en su buffer interno, los mensajes de un módulo específico hasta que los módulos remotos se los soliciten. Asimismo, para que se pueda producir la comunicación entre coordinador de sueño y módulos remotos el parámetro SP de ambos deberá ser seteado con el mismo valor.

Modo Comando: Permite configurar y ajustar ciertos parámetros de los módulos Xbee mediante el envío de comandos AT. Para ello se puede utilizar el emulador de terminal GTKTerm o el programa XCTU de Digi International. La sintaxis de un comando AT para, por ejemplo, establecer el pin número 11 (DIO4) como entrada digital es la siguiente:

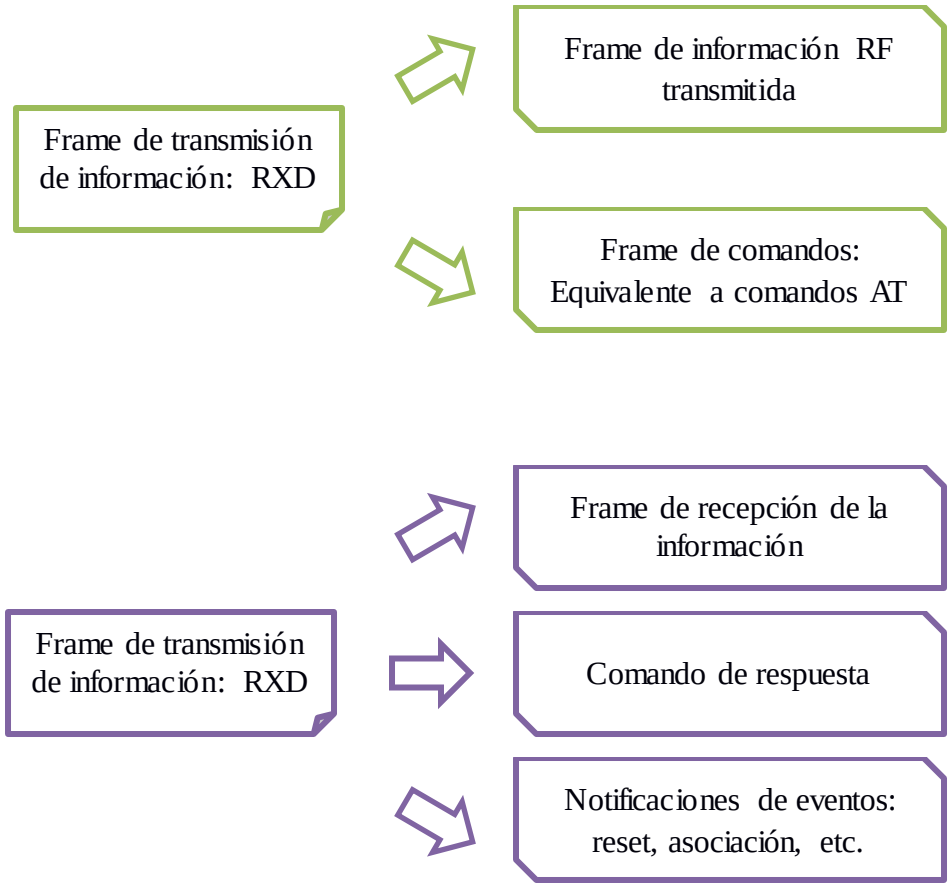
ATD43<CR>

AT	**D4**	**3**	**<CR>**
Prefijo AT	Comando ASCII	Parámetro hexadecimal	Retorno de carro

[Tabla 1] Ejemplo de comando AT

Modo Transparente: Es el que viene por defecto en los módulos Xbee y se utiliza cuando se necesita una conexión punto a punto sin ningún tipo de control. En dicho modo los pines 2 (RXD) y 3 (TXD) son utilizados por el módulo para recibir y transmitir los paquetes de información. Si estos son recibidos por el pin RXD serán guardados en el buffer de entrada y empaquetados (integrados en un paquete de radiofrecuencia). A partir de este momento el módulo comprobará si el buffer se encuentra lleno o si se ha cumplido el tiempo establecido en el comando RO sin que entre ningún paquete nuevo. Desde el momento en el que se cumple alguna de estas dos condiciones se podrá transmitir la información.

Modo API: Permite que toda la información que entre y salga del módulo Xbee sea empaquetada en frames y que estos definan las operaciones y eventos que sucederán en el interior del dispositivo. Estos, además, disponen de cabeceras que aseguran la entrega del mensaje al más puro estilo TCP (Protocolo de Control de Transmisión), uno de los protocolos más importantes de Internet. Existen dos tipos de frames, los que trasmiten información y los que la reciben:



[Fig. 4] Tipos de frames

Por lo tanto, el modo de operación API sirve para que un cliente pueda configurar uno o varios módulos Xbee de manera alternativa al modo comandos, identificar la dirección de origen de cada paquete recibido y recibir el estado de éxito o fallo de cada paquete RF transmitido entre otras cosas.

Modo IDLE: Cuando el módulo no se encuentra ni transmitiendo, ni recibiendo, ni en modo comando, ni ahorrando energía, es decir, cuando no se encuentra en ninguno de los otros modos.[11]

2.1.5. Modos de configuración

Los módulos Xbee los podemos configurar para que operen de diferentes maneras, centrándonos en este apartado en el modo de conexión API, ya que es el que resulta relevante durante la implementación. Para configurar el modo de operar del dispositivo se debe establecer una determinada versión de firmware. Existen cuatro tipos que harán comportarse de una manera u otra a nuestros módulos. Para hacer que los módulos trabajen en modo API elegiremos los firmware 1.1.XX o 1.3.XX según queramos que funcionen como coordinador o como router/end device.

- **Firmware `1.0.xx`:** Define al módulo Xbee como coordinador trabajando en el modo de operación transparente.
- **Firmware `1.1.xx`:** Define el dispositivo como coordinador trabajando en el modo de operación API.
- **Firmware `1.2.xx`:** Define al Xbee como Router o End Device y lo hace trabajar en modo transparente.
- **Firmware `1.3.xx`:** Define al módulo como Router o End Device y lo hace operar en el modo API.

El modo de conexión API permite enviar los paquetes de información (utilizando comandos AT) de forma estructurada y definida, permitiendo que unos módulos modifiquen parámetros a otros o que simplemente comprueben el estado de sus vecinos. El envío o la recepción de las tramas API se realizan a través del uso de frames de datos UART (Transmisor/Receptor Asíncrono Universal), siendo localizable el origen de cualquier paquete recibido dentro de la red. También existe la posibilidad de verificar, mediante el Checksum, que los datos entregados son exactamente iguales a los datos enviados, es decir, comprueba que los datos no se hayan corrompido por el camino. [14]

Estructura de un frame en modo API:

Delimitador inicial	**Longitud**		**Frame**	**Checksum/CRC**
0x7E	Bits más significativos	Bits menos significativos	Estructura API	¿Paquetes corrompidos?

Tamaño:

1 byte	**2 ó 3 bytes**	**Mínimo 4 bytes**	**1 byte**

[Tabla 2] Estructura y tamaño de un frame en el modo de conexión API

Ejemplo de estructura API: Mensaje para establecer el pin número 11 (DIO4), de un módulo Xbee remoto, como entrada digital.

0x17	0x00	0x00	0x13	0xA2	0x00	0x40	0x34	0x0E	0x6A	0xFF	0xFE
ID trama API		Dirección de 64 bits								Dir. de 16 bits	

0x02	0x44	0x34	0x03
Opción del comando	Comando AT: **D4**		Valor del parámetro: **3**

[Tabla 3] Ejemplo de estructura API

El primer byte de la trama anterior (0x17) es un **identificador** y se refiere al tipo de mensaje API que contiene la trama de datos que le sigue. Los identificadores de tramas API existentes son los siguientes:

Valor	Nombre del identificador
0x8A	Estado del modem
0x08	Comando AT
0x09	Comando AT: Enviar a la cola el valor del parámetro
0x88	Respuesta a comando AT
0x17	Solicitud de comando remoto
0x97	Respuesta a comando remoto
0x10	Solicitud de transmisión Zigbee
0x11	Trama de comando para direccionamiento explícito ZB
0x8B	Estado de transmisión Zigbee
0x90	Recepción de paquete Zigbee [AO = 0]
0x91	Indicador de Rx explícito Zigbee [AO = 1]
0x92	Indicador de muestreo de datos Rx de E/S Zigbee
0x94	Indicador de lectura de sensor Xbee [AO = 0]
0x95	Indicador de identificación de nodo [AO = 0]

[Tabla 4] Identificadores de trama API

En esta memoria se mostrará el funcionamiento específico de dos identificadores de trama utilizados durante la implementación. Primero, el referido a la solicitud de un comando remoto, el 0x17. Segundo, el relacionado con el muestreo de las entradas y salidas, el 0x92.

Identificador de trama 0x17: Permite enviar un comando AT a un módulo remoto. En el caso que nos ocupa ha sido utilizado para enviar órdenes a nuestro actuador, permitiendo activarlo y desactivarlo a nuestra conveniencia.

ESTRUCTURA API ESPECÍFICA PARA: 0x17

Identificador API	0x17	
Datos del identificador especificado	Frame ID - Byte número 5	Si se establece a 0 se anula la respuesta automática a un comando AT.
	Dirección de destino: 64 bits - Bytes del 6 al 13	MSB (Bits más significativos) primero. LSB (Bits menos significativos) después.
	Dirección de destino: 16 bits - Bytes del 14 al 15	Se pone 0xFFFF si la transmisión es abierta o si se desconoce la dirección.
	Opciones del comando - Byte número 16	Enviar el byte 0x02 para que los cambios se apliquen en el dispositivo remoto.
	Nombre del comando - Bytes del 17 al 18	Comando a enviar
	Parámetro del comando - Byte número 19	Establece un parámetro en el nodo remoto. Si no se coloca nada significa que se pretende hacer una consulta. (P ej. El comando IS).

[Tabla 5] Estructura API específica para el identificador de trama 0x17

Identificador de trama 0x92: Permite comprobar el estado de las entradas y salidas de un dispositivo Xbee remoto. En nuestro caso utilizaremos el comando AT *IS* para ello.

ESTRUCTURA API ESPECÍFICA PARA: 0x92

Identificador API	0x92	
Datos del identificador especificado	Dirección de destino: 64 bits - Bytes del 5 al 12	MSB (Bits más significativos) primero. LSB (Bits menos significativos) después.
	Dirección de destino: 16 bits - Bytes del 13 al 14	Se pone 0xFFFF si la transmisión es abierta o si se desconoce la dirección.
	Opciones de recepción - Byte número 15	Byte 0x01 para que verifique la recepción del paquete. Byte con 0x02 si se trata de una simple difusión.
	Número de muestras - Byte número 16	Siempre se establece a 1. Número de muestras incluidas en la carga útil.
	Máscara de canales digitales - Bytes del 17 al 18	Indica que canales digitales (E/S) del dispositivo remoto han permitido el muestreo.
	Máscara de canales analógicos - Bytes del 19 al 20	Indica que canales analógicos del dispositivo remoto han permitido el muestreo.
	Muestras de E/S digitales* - Bytes del 20 al 21	Resultados del muestreo de las E/S digitales habilitadas. Ocupará 2 bytes en caso de comprobarse todos los canales digitales.
	Muestras de entradas analógicas - Bytes del 22 al 23	Resultados del muestreo de las entradas analógicas. Están clasificadas secuencialmente desde AD0/DIO0 hasta AD3/DIO3

[Tabla 6] Estructura API específica para el identificador de trama 0x92

Máscaras de E/S digitales*: La posición de los canales digitales en la máscara de bits seguirá un orden secuencial al nombre de cada canal y nunca secuencial a la ubicación del pin en el dispositivo. A continuación, y a modo de ejemplo se muestra una tabla con los seis primeros canales digitales. [14]

Posición en la máscara de bits	6	5	4	3	2	1	0
Canal digital	DIO 6	DIO 5	DIO 4	DIO 3	DIO 2	DIO 1	DIO 0
Pin utilizado	16	15	11	17	18	19	20

[Tabla 7] Distribución de posiciones en la máscara de bits para canales digitales

2.2. Arduino

2.2.1. ¿Qué es Arduino?

Arduino nació en Italia en el año 2005 y comenzó como un proyecto para los estudiantes del *Instituto de diseño interactivo Ivrea*. Su nombre proviene del *Bar di Re Arduino* (Bar del Rey Arduino), establecimiento que rinde homenaje a un antiguo monarca italiano y en el que uno de los fundadores del proyecto (Massimo Banzi) pasaba algunas horas.

En una explicación sencilla diríamos que Arduino consiste en una familia de placas electrónicas con microcontrolador, capaces de ser programadas y de ejecutar las órdenes grabadas en su memoria. Sin embargo, Arduino no es solo eso, también es una plataforma *open source* (código abierto) con un entorno de desarrollo integrado (IDE) que soporta lenguajes de programación como C y C++ y que está diseñado para facilitar el uso de la electrónica en proyectos de múltiples disciplinas. Su conjunto de entradas y salidas analógicas y digitales permiten controlar todo tipo de actuadores y sensores, acoplar tarjetas de expansión (escudos) y relacionarse con otros circuitos. Además, las placas cuentan con interfaces de comunicación serie, USB en la mayoría de modelos, que permite cargar los programas desde un ordenador personal sin dificultad alguna.

Existe una gran variedad de placas con diferentes características como la tensión utilizada (3,3 ó 5 V dependiendo del microcontrolador empleado), el número de entradas y salidas, el procesador instalado, la memoria, la posibilidad de alimentar otros elementos desde la placa o la capacidad para poder conmutar el voltaje.

2.2.2. Características de la versión utilizada

En este trabajo fin de grado se ha trabajado con la placa ***Arduino Nano v3***, modelo elegido por su simplicidad, bajo peso y dimensiones. Tiene una funcionalidad parecida a la Arduino Duemilanove y se alimenta mediante la conexión mini USB de tipo B, a través del pin número 30 (Vin) con una fuente de alimentación no regulada o utilizando una fuente regulada en el pin número 27.

Especificaciones	
Microcontrolador	ATmega328
Voltaje de operación	5 V
Voltaje de entrada recomendado	7 – 12 V
Límites para el voltaje de entrada	6 – 20 V
Pins de entradas y salidas digitales	14 (6 PWM)
Pins de entradas analógicas	8
Corriente AC por E/S	40 mA
Memoria Flash	32 Kb
SRAM	2 Kb
EEPROM	1 Kb
Frecuencia de reloj	16 MHz
Largo	45 mm
Ancho	18 mm
Peso	5 g

[Tabla 8] Especificaciones de Arduino Nano

Como ya comentamos anteriormente, la placa tiene un total de 8 entradas analógicas (distribuidas a su derecha) y 14 entradas/salidas digitales (distribuidas a su izquierda), de las cuales 6 de ellas pueden funcionar como PWM. Asimismo, el pin número 27 tiene una segunda funcionalidad, puede ser utilizado para alimentar a 5 voltios otros dispositivos.

TX	1	30	Vin
RX	2	29	GND
Reset	3	28	Reset
GND	4	27	5 V
D2	5	26	A7
D3	6	25	A6
D4	7	24	A5
D5	8	23	A4
D6	9	22	A3
D7	10	21	A2
D8	11	20	A1
D9	12	19	A0
D10	13	18	Ref
D11	14	17	3V3
D12	15	16	D13

[Tabla 9] Patillaje de Arduino Nano

Donde:

- **D?:** Va referido a entradas o salidas digitales.
- **A?:** Va referido a entradas analógicas.
- **TX:** Pin de transmisión.
- **RX:** Pin de recepción.

2.3. GNU/Linux

2.3.1. ¿Qué es GNU/Linux?

GNU/Linux es un término utilizado para referirse a la combinación de **GNU**, sistema operativo, y de **Linux**, núcleo (kernel) libre similar a Unix. En este sentido, la función de un **núcleo** o **kernel**, es facilitar a los distintos programas el acceso seguro y ordenado al hardware del equipo, encargándose a su vez de la gestión de los recursos. Linux, como el resto de los núcleos, garantiza de esta forma la ejecución de los procesos y propone al mismo tiempo una interfaz entre el espacio dedicado al núcleo y los programas del espacio dedicado al usuario. De la misma manera, un **sistema operativo** es simple y llanamente un conjunto de programas fundamentales que un equipo informático necesita para poder comunicar y recibir instrucciones de los usuarios.

El proyecto GNU (GNU is not Unix) se anunció públicamente en el año 1983 y tuvo como objetivo el crear un sistema operativo completamente libre. Esto es que se permite la copia, el estudio, la modificación y la utilización con cualquier fin. Además, el término libre también permite la redistribución con o sin cambios o mejoras. De esta manera, Richard Stallman comenzó con su ensayo "Manifiesto GNU" a establecer sus motivaciones para realizar el proyecto, destacando entre ellas el querer volver al espíritu de cooperación que prevaleció en los tiempos iniciales de la comunidad de usuarios de ordenadores. El proyecto progresó con el liderazgo del Stallman y el apoyo de académicos y programadores voluntarios. Consiguieron crear el primer sistema operativo completamente libre, lograron proteger su libertad para ejecución, copiado, modificado y distribución gracias a la idea de copyleft contenida en la Licencia General Pública de GNU (GPL).

Hacer que el sistema GNU fuera compatible con la arquitectura UNIX (sistema operativo **no libre**) implicaba beneficiarse de pequeñas piezas individuales de software, muchas de las cuales, como el sistema gráfico X-Windows o el editor de textos TeX, ya estaban disponibles. No obstante, otros software tuvieron que reescribirse.

El núcleo Linux, por otro lado, fue concebido en el año 1991 por el finlandés Linus Torvalds, entonces estudiantes de ciencias de la computación en Helsinki. En sus inicios comenzó intentando obtener un kernel similar a Unix y que funcionara con los microprocesadores de Intel 80386. Al poco tiempo muchas personas se unieron al proyecto y ayudaron con el código, logrando así sacar la primera versión de Linux. El núcleo, al igual que GNU, fue distribuido bajo la licencia GPL, hecho que permitió a GNU continuar con el desarrollo del proyecto al incorporar el kernel de Torvalds. Este sustituyó al núcleo que Stallman y su equipo estaban desarrollando, el llamado Hurd, ya que aun no era lo suficientemente maduro. Nació así GNU/Linux, siguiendo la tradicional filosofía de cooperación entre desarrolladores.

2.3.2. Distribuciones GNU/Linux

Una **distribución** o **distro** es un conjunto de software específico que se encuentra ya compilado y configurado. Una **distro** de **GNU/Linux** es una distribución de software basada en el kernel Linux que incluye herramientas del proyecto GNU. Estas pueden ser herramientas administrativas, procesadores de textos, hojas de cálculo, reproductores multimedia o el propio sistema de ventanas X-Windows entre otras. Las distribuciones pueden estar mantenidas por la comunidad, como Debian, mantenidas por una empresa, como Canonical Ltd con Ubuntu o simplemente no estar relacionadas con ninguna empresa o comunidad, como en el caso de Slackware. Asimismo, las diferentes distros pueden clasificarse de varias maneras: por la distribución de la que deriven, por el gestor de paquetes utilizados (deb, rpm, etc), por la inactividad de su desarrollo o incluso por el propósito al que estén destinadas (sistemas empotrados, videojuegos, educativas, servicios de redes, etc).

Los **sistemas de gestión de paquetes** sirven para automatizar y facilitar el proceso de instalación, actualización, configuración y eliminación de paquetes de software. Esto es que tienen la tarea de organizar todos los paquetes del sistema y, además, se encargan de su usabilidad. No se debe confundir un gestor de paquetes con un instalador, ya que ambos se diferencian por características como las siguientes:

- Sistema de Gestión de Paquetes
    - Forma parte del sistema operativo.
    - Puede verificar y administrar todos los paquetes sobre el sistema.
    - Solo existe un único formato de paquetes.
    - Solo existe un único vendedor de sistema de administración de paquetes.

  - Usa una única base de datos de instalación.
- Instalador
  - Cada producto viene unido a su propio instalador.
  - Solo trabaja con su propio producto.
  - Existen múltiples formatos de instalación.
  - Existen múltiples vendedores de instalador.
  - Rastrea su propia instalación.

2.3.3. Distribución utilizada: Debian GNU/Linux

En el trabajo fin de grado actual se ha utilizado la distribución de GNU/Linux ***Debian 8.0 Jessie***, lanzada en abril del año 2015. El proyecto Debian es llevado a cabo por una gran comunidad de desarrolladores voluntarios (actualmente más de mil) que se encuentran repartidos alrededor del mundo y que realizan su trabajo mediante colaboraciones a través de Internet. Cada uno de ellos tiene una responsabilidad u ocupación dentro del proyecto Debian, ya sea relacionado con la infraestructura del mismo (coordinación de lanzamientos, traducciones de web, etc) o relacionado con los paquetes (mantenimiento, documentación, control de calidad). El proyecto en sí es una organización voluntaria con tres documentos fundadores:

- **El contrato social de Debian:** Define las bases por las cuales el proyecto y sus desarrolladores tratan los asuntos.

- **Las directrices de software libre de Debian:** Definen los criterios del Software libre y dictan qué software es aceptable para la distribución según lo referido al contrato social.

- **La constitución de Debian:** Describe la estructura de la organización para la toma de decisiones de manera formal dentro del proyecto.

Hasta la fecha se han lanzado trece versiones de Debian y como curiosidad a destacar, los nombres de estas han sido cogidos de la película de animación Toy Story. Las primeras diez versiones carecen actualmente de soporte por ser distribuciones antiguas. No obstante, las tres últimas versiones siguen teniendo soporte a día de hoy. Si seguimos la trayectoria de desarrollo es probable que para el año 2017 tengamos una nueva versión de esta distro, la conocida con el nombre en clave ***Stretch***.

2.4.Ordenadores de placa reducida

2.4.1. ¿Qué es un ordenador de placa reducida?

Expresado de manera simple, las placas miniordenadores son ordenadores completos reducidos en un sólo circuito. Estos no poseen tarjetas de expansión que provean al equipo de puertos seriales, controladores para discos duros, gráficos o sonido, sino que en la misma placa base tienen un microprocesador, entradas, salidas, memoria RAM y el resto de características que un ordenador funcional necesita. Los ordenadores de placa reducida deben su tamaño y su ligereza a los grandes niveles de reducción e integración de componentes y conectores que existe actualmente. Además, manejan mejor la potencia eléctrica que los ordenadores de múltiples tarjetas.

La principal desventaja surge cuando queremos actualizar o ampliar uno de estos sistemas. Normalmente resulta imposible. Si se diera un fallo en alguno de los componentes tocaría desechar la placa completa y conseguir una nueva, cosa que no sucede con los ordenadores comunes. La filosofía modular de estos nos permite sustituir los componentes que dejen de funcionar sin tener que perder todo el ordenador. Por ejemplo, podríamos sustituir la memoria RAM si esta fallara.

En el mercado actual existen numerosas empresas que distribuyen ordenadores de placa reducida, las cuales presentan diferencias en la velocidad y el tipo de procesador que poseen, en la capacidad de la memoria RAM o en los puertos que incluyen (USB, pines GPIO, HDMI, Ethernet, etc). Entre las placas a destacar se encuentran *Beaglebone*, *Raspberry Pi*, *Pine A64*, *Banana Pi* o la recién llegada *Arduino Tian* entre otras. Todas ellas compiten entre sí presentando características y precios bastante competitivos y asequibles. A continuación, y para entender sus diferencias, se muestra una pequeña tabla comparativa con modelos que son similares entre sí. No obstante, resaltar que aquí se hará especial hincapié en la más popular entre la comunidad de desarrolladores, la **Raspberry Pi**.

	Raspberrry Pi	**Beaglebone**	**Pine A64**	**Banana Pi**	**Arduino**
Modelo	B	Black	PINE64	Banana Pi	Tian
Procesador	700 MHz	1000 MHz	1.2 GHz	1000 MHz	535 MHz
Núcleo	Monocore	Monocore	Dual core	Dual Core	Monocore
RAM	512 Mb	512 Mb	512 Mb	1024 Mb	64 Mb
Puertos USB	2	1	2	2	1
Pines GPIO	26	96	46	26	20+7
Ethernet	100 Mbit	100 Mbit	100 Mb	Gigabit	

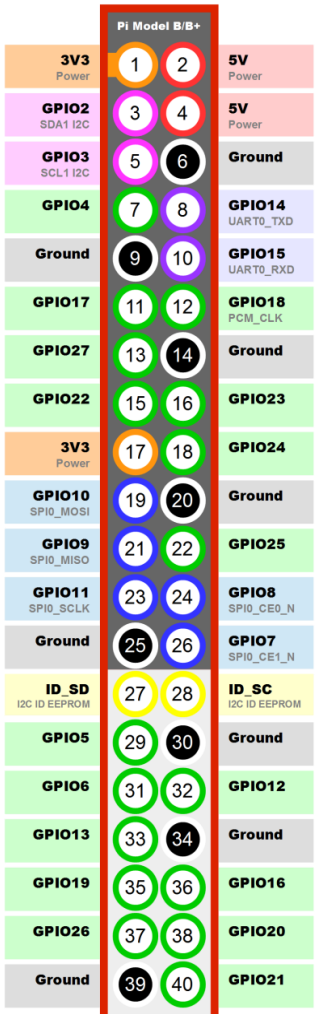
[Tabla 10] Comparativa de distintos ordenadores de placa reducida

Las aplicaciones de estas placas mini-ordenadores son muy amplias y diversas. Para empezar el llevar un ordenador del tamaño de la palma de tu mano, he incluso más pequeño y ligero con la nueva Raspberry Pi Zero, abre muchas posibilidades que hasta hace unos años no se consideraban. Los usos de una de estas placas van desde convertir cualquier pantalla en un ordenador, hasta ser utilizadas en proyectos del ámbito de la robótica. La comunidad de desarrolladores, expertos y aficionados, que se ha creado en torno a ellas ha creado proyectos de todo tipo, como por ejemplo mini-estaciones meteorológicas, mediacenters caseros, aplicaciones domótica, consolas retro o incluso un teléfono móvil totalmente operativo y funcional, el llamado PiPhone.

2.4.2. Raspberry Pi

La opción elegida para la implementación del trabajo desarrollado ha sido Raspberry Pi, principalmente por la gran comunicada que existe detrás y el respaldo que esta ofrece a los sistemas operativos que permite ejecutar la placa. Asimismo, esta ofrece unas prestaciones suficientes para desarrollar el trabajo implementado.

Este ordenador de placa reducida nació con el objetivo de animar a los niños a aprender informática en las escuelas y su desarrollo se llevó a cabo en el Reino Unido gracias a la Fundación Raspberry Pi y a su administrador, Eben Upton. Este proceso se llevó a cabo con los primeros diseños durante el año 2006 y posteriormente con la creación de la fundación en 2009. No obstante, no fue hasta seis años después del origen del proyecto, año 2012, que se pusieran a la venta las primeras placas. Vendieron quinientas mil unidades durante los primeros seis meses posteriores al lanzamiento oficial.



[Fig. 5] GPIO de Raspberry Pi

Si nos centramos en el hardware, la propiedad de estas tarjetas se encuentra registrada, en cambio su uso es completamente libre. Esto quiere decir que cualquiera puede convertirse en revendedor o redistribuidor de las placas Raspberry Pi. Por lo general, su hardware cuenta con un microprocesador, puertos USB, Ethernet, memoria RAM, entradas y salidas de vídeo, salidas de audio, almacenamiento integrado (mediante tarjetas SD), etc. Además tiene un

conjunto de pines de entrada y salida de propósito general (GPIO) para permitir la programación (principalmente en el lenguaje Python) de los mismos por parte de los usuarios.

Por otro lado, el software oficial es open source, ya que utiliza una versión adaptada de Debian, la denominada **RaspBian**. También se permite usar otros sistemas operativos, algunos de ellos con soporte oficial para las descargas por parte de la fundación como las distribuciones Pidora (derivado de Fedora) y Arch Linux ARM (derivado de Arch Linux) o la propia Raspbian (derviada de Debian). Otros, en cambio, tienen un desarrollo y soporte exclusivo de la propia comunidad. Entre estos podemos destacar Minibian (versión minimalista de Raspbian), Kano (enfocado a los niños, incluye software educativo) y Retropie (emulador de consolas antiguas).

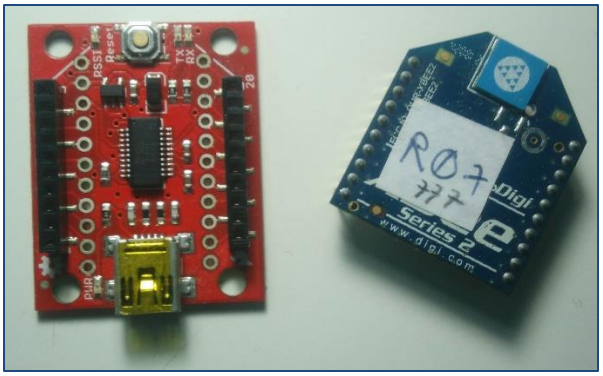
3. IMPLEMENTACIÓN

En las siguientes líneas se expresará el trabajo desarrollado en el presente TFG. Como bien se ha dicho en la introducción, este trata de la implementación de un sistema domótico inalámbrico enfocado a la creación y configuración de un sistema de alarma portátil con la ayuda de Xbee, Arduino y un ordenador personal.

3.1.Montaje del circuito desarrollado

El montaje desarrollado para este sistema de alarma portátil consta de dos partes. La primera contiene un módulo Xbee, configurado como Coordinador, y un ordenador personal corriendo un sistema GNU/Linux. La segunda parte de este montaje tendrá otro módulo Xbee, configurado como nodo remoto, un dispositivo Arduino Nano, un sensor de ultrasonidos HC-SR04 y un buffer o zumbador que emitirá una serie de pitidos cuando la alarma sea activada. Por lo tanto:

Circuito A: Es el encargado de controlar, coordinar y configurar todo el sistema. También se encarga de avisar al usuario mediante un correo electrónico cuando la alarma ha sido activada. El circuito solo está compuesto del equipo informático y el módulo Xbee. Para interconectar ambos se utilizará la placa Xbee USB Explorer [5] que permite conectar y utilizar cualquier dispositivo Xbee directamente mediante un puerto USB. Esta placa está prevista de dos circuitos integrados para que funcione correctamente. El primero de ellos es el MIC5219 [9], que permite ajustar el voltaje de alimentación a los 3.3 V que soporta nuestro Xbee. El segundo circuito es el que nos proporciona la comunicación serial entre nuestros dispositivos. Se trata del FT231X [6] y se podría considerar como un adaptador USB-UART. Nos permite acceder mediante una conexión USB a los pines UART de nuestro módulo Xbee, con tasas de transferencia que van desde los 300 baudios hasta los 3 Mbaudios.

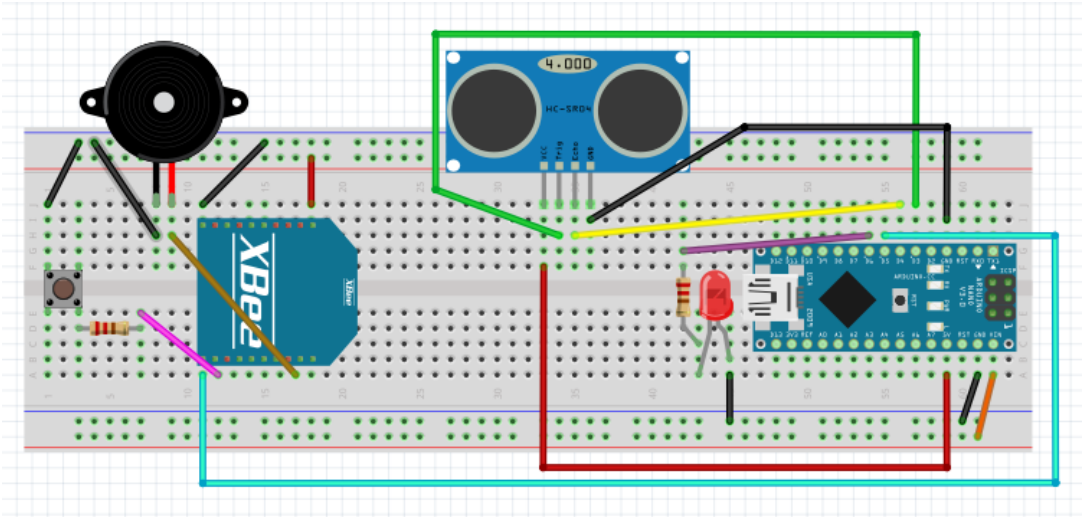


[Fig. 6] Xbee USB Explorer

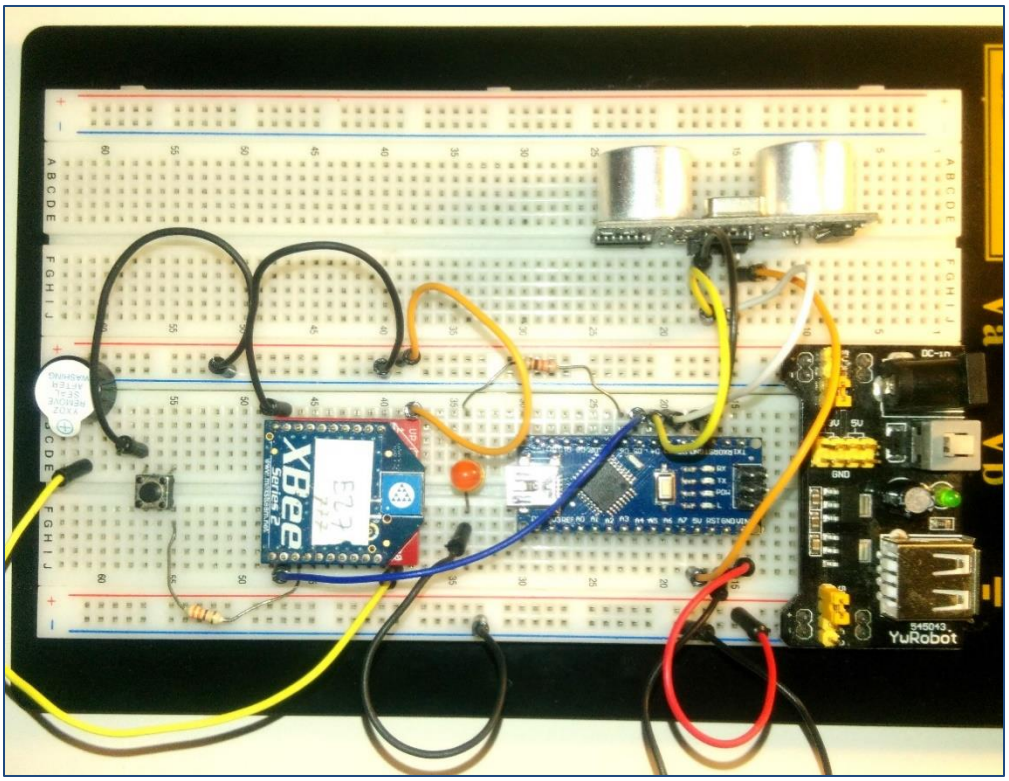
- Módulo Xbee Coordinador: Estará en comunicación continua con el otro módulo Xbee. Una de sus funciones principales será comprobar constantemente si ha recibido algún mensaje en su buffer de entrada. Si esto sucede, el equipo informático al que se encuentre conectado le proporcionará una serie de órdenes para actuar en consecuencia transmitiéndolas al nodo Xbee ubicado en el circuito B. La conexión con el ordenador personal será serial y a través de un puerto USB. Para ello se utilizará la placa Xbee USB Explorer de Sparkfun antes mencionada.

- Equipo informático: Sea un ordenador personal corriendo un GNU/Linux como Debian o una Raspberry Pi ejecutando un Raspbian como sistema operativo, la misión principal de este equipo será disponer de los elementos necesarios para realizar las comunicaciones entre módulos Xbee y coordinar toda la red. Para ello realizará tareas de control de todo el sistema (permitiendo su configuración por parte del usuario) ejecutando de manera continua (funcionamiento demonio) un programa escrito en el lenguaje de programación C++. Esto quiere decir, que ante las variaciones producidas en el circuito B el programa demonio deberá evaluar esos cambios y, en consecuencia, proporcionarle a ese circuito una respuesta apropiada. Además, debe notificar al usuario (por correo electrónico) que ha recibido un aviso para activar la alarma y ha enviado la orden.

Circuito B: Tiene como misión ejecutar las órdenes que le llegan desde el circuito A y avisar a este cuando se produzcan variaciones en las entradas del módulo Xbee remoto.



[Fig. 7] Circuito B: Montaje en protoboard

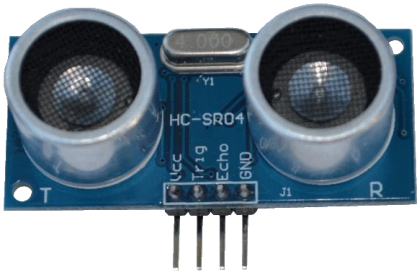


[Fig. 8] Circuito B: Montaje real en protoboard

- Módulo Xbee remoto: Trabajará como End Devices ya que no tiene otros nodos con los que comunicarse. Es el puente de comunicación existente entre los sensores/actuadores y el circuito A. Se encarga de recibir la información del sensor y enviar órdenes al actuador. Para el primero recibe la información de forma indirecta, ya que el pequeño dispositivo Arduino se encarga de realizar las mediciones y activar o no uno de los pines de entrada del módulo Xbee cuando se den determinadas circunstancias en dichas medidas. En el segundo caso, el Xbee pondrá en estado lógico alto (activando así el actuador) el pin que tenga definido como salida digital siempre y cuando reciba esta orden del Xbee Coordinador.

- Arduino Nano: Su misión es controlar de una forma más eficiente el sensor de ultrasonidos HC-SR04. Será por tanto el encargado de activar la señal de ping y verificar la recepción de la misma en el pin Eco de dicho sensor. Cuando el sensor de a Arduino una distancia inferior a la definida previamente, el dispositivo pondrá en estado lógico alto una de sus salidas digitales. Esto supondrá un aviso para módulo Xbee Remoto, ya que uno de sus pines de entrada digital será activado.

- Sensor Ultrasonidos HC-SR04: Sirve para medir distancias. Funciona enviando ultrasonidos (a 340 m/s en el aire) a través de un transductor, es decir, por uno de los cilindros que componen el sensor. Seguidamente el dispositivo espera que dicha onda rebote sobre un objeto y vuelva para que sea captada por el otro cilindro. Su rango de distancias va desde los 3 centímetros hasta los 3 metros, con una precisión de 3 milímetros.



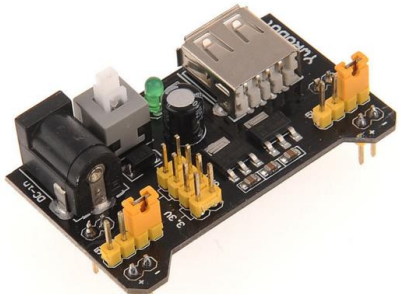
[Fig. 9] Sensor ultrasonidos HC-SR04

- Zumbador/Buzzer: Dispositivo que transforma la electricidad en sonido (transductor electroacústico). Esto lo consigue permitiendo a la corriente eléctrica pasar por la bobina de un pequeño electroimán para que este produzca un campo magnético variable que haga vibrar una lámina de acero sobre su armadura. En el presente proyecto este *buzzer* hará las funciones de una alarma al emitir un zumbido cuando el Xbee Coordinador envíe la orden de activación al Xbee Remoto.

[Fig. 10] Buzzer/Zumbador

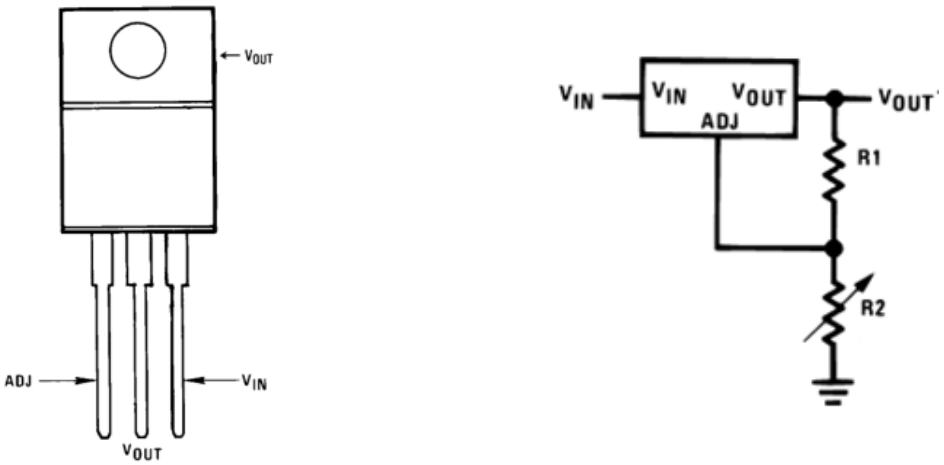
Alimentación del circuito B: Para alimentar este circuito es muy práctico utilizar un módulo de alimentación como el MB102 [4], ya que nos permite alimentar, mediante una conexión USB, dos dispositivos a distinto voltaje. Hay dos maneras de suministrar energía a este módulo. La primera es proporcionándole 5 V mediante la a través de la conexión USB, la segunda es suministrándole un voltaje de entre 6.5 V a 12 V (DC) mediante una fuente de alimentación. Para alimentar los dispositivos electrónicos de nuestro circuito nos podemos valer de sus dos salidas de tensión fija: la primera a 3.3 V alimentará a nuestro módulo Xbee, la segunda a 5 V proveerá de energía a nuestra placa Arduino Nano. La corriente máxima que pueden prestar estas salidas es de 700 mA, suficiente para hacer trabajar nuestro circuito. Para seleccionar el

voltaje de cada salida disponemos de dos puentes (jumpers). Por otro lado, el funcionamiento de este módulo de alimentación está basado en el uso del regulador lineal de tensión A1117 [1], que se compone de algunos módulos que incluyen un circuito de arranque, un circuito de polarización, un apagado térmico y un limitador de corriente entre otros. Existen dos modelos de este regulador, el primero permite ajustar su voltaje de salida y el segundo tiene la tensión de salida fija. En el caso de nuestro módulo MB102 se utiliza la versión del regulador que proporciona la tensión de salida fija.



[Fig. 11]
Módulo de alimentación MB-102

Cabe destacar que utilizar el módulo de alimentación contectado a una toma de corriente limita en cierto modo la portabilidad de nuestro sistema de alarma inalámbrico. Por tanto se pueden explorar otras opciones. Una de ellas sería complementar nuestro módulo MB102 con baterías externas con conexión USB. Otra de las altenativas sería deshacernos de nuestro módulo MB102 y optar por fabricar dos circuitos de alimentación. El primero para proveer una tesión fija 3.3 V y el segundo para suministrar 5 V. Estos circuitos los construiriamos por separado, se alimentarían de pilas eléctrias y se utilizaría un circuito integrado *LM317* [8] en cada uno de ellos. Para obtener la salida de tensión constante añadieremos dos resistencia de valor ya predeterminado al LM317, de esta forma construiremos un partidor de tensión que nos proporcionará el voltaje fijo. También existe la posibilidad de que la tensión saliente sea regulable si cambiamos una de las resistencias fijas por otra variable (potenciómetro). En nuestro caso no sería necesario porque deseamos mantener los voltajes salientes a 3.3 V y a 5 V.

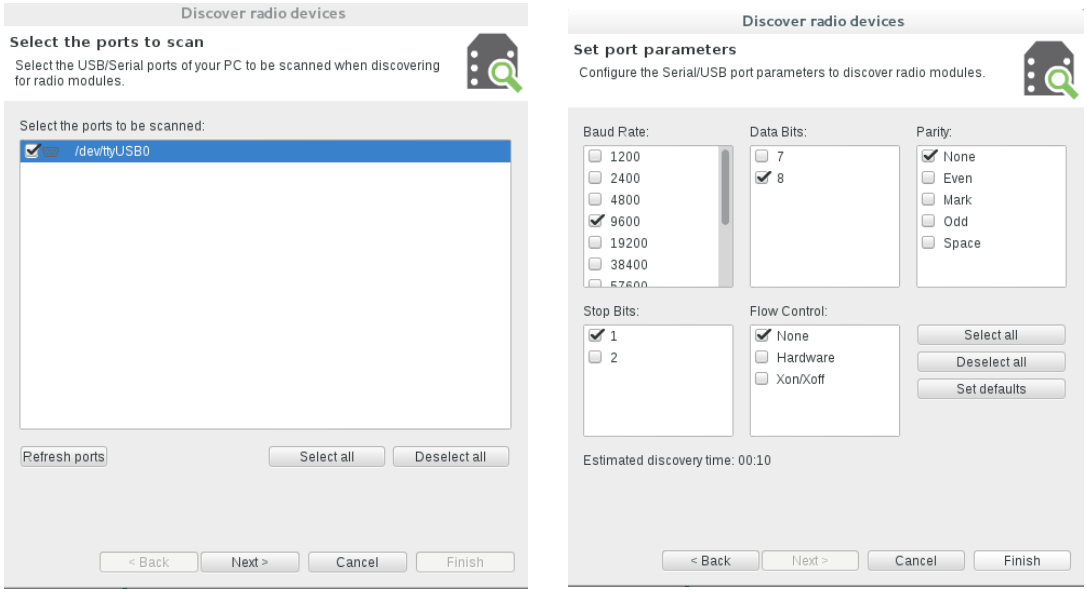


[Fig. 12] Izquierda: Circuito integrado LM317 (vista frontal).
Derecha: Partidor de tensión con resistencia variable

3.2.Configuración de los módulos Xbee

La configuración inicial de los dispositivos de Digi International se puede realizar de manera sencilla con el software X-CTU que la propia compañía distribuye. En la web *www.digi.com/es/* podremos encontrar versiones del mismo disponibles para Microsoft Windows, Mac OS y GNU/Linux. El software nos permitirá configurar una multitud de parámetros de nuestros dispositivos, entre los que cabe destacar la configuración propia los pines (definirlos como entradas analógicas, como entradas o salidas digitales, etc), definir el modelo de Xbee utilizado, el modo de operación del módulo y la actualización del firmware entre otros. Además, X-CTU nos permitirá establecer comunicación entre nuestros módulos mediante el envío de comandos AT o de tramas API.

Para configurar nuestros módulos debemos primero hacer que el software de Digi los encuentre. Lo haremos ejecutando la opción *Discover radio modules* de la pestaña *XCTU*. Tras seleccionar el puerto /dev/ttyUSB0 al que se encuentra conectado nuestro dispositivo tendremos la posibilidad de modificar determinadas opciones como la paridad, los bits de datos o la velocidad a la que se establece la conexión (baudius rate). Por lo general, es suficiente con revisar la velocidad y dejar el resto de parámetros con el valor que vienen por defecto.



[Fig. 13] XCTU: Discover radio devices

Localizado el primer módulo lo configuraremos para que sea el Zigbee Coordinador. Para ello hacemos click en el icono del Xbee situado a la izquierda de la ventana y nos aparecerá la ventana de ajustes. En ella podremos encontrar todos los parámetros configurables de nuestro dispositivo: entradas, salidas, nombre, PAN ID, etc. Haciendo click en el icono de *Update*

Firmware podremos actualizar firmware del dispositivo definiendo la versión deseada, el modelo de Xbee utilizado y el modo de operación del módulo. En nuestro caso seleccionaremos el firmware 11.XX, que es el encargado de decirle al módulo que opere como coordinador y en modo API. Esto nos permitirá comunicarnos con el Xbee End Device mediante tramas API.

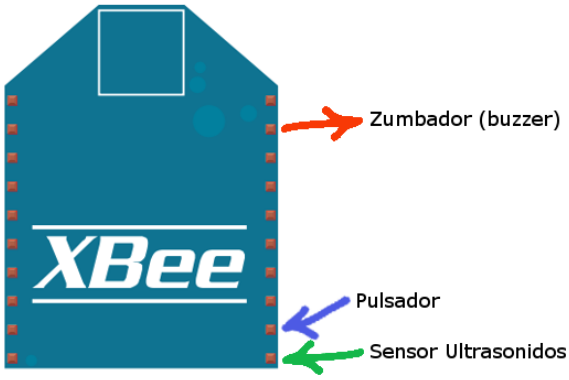
- Modelo: XB24-B
- Modo de operación: ZNet 2.5 Coordinador API
- Versión de Firmware: 1147

Además, si queremos que los módulos se encuentren el uno al otro debemos configurar en la ventana de ajustes el mismo valor ID para la red PAN en cada uno de los módulos. Su rango va desde 0x0 hasta 0xFFFF y también se puede configurar a través del comando ID. En nuestro caso el valor se corresponderá con 1234. Conjuntamente, y para evitar problemas en un futuro, debemos asegurarnos de que hemos fijado el *Baudios Rate* (velocidad de conexión del dispositivo) para todos nuestros módulos Xbee. En el caso que nos ocupa la velocidad será de 9600 baudios por segundo. Terminada la configuración inicial podemos grabar los cambios en el icono denominado *Write radio setting*.

Configurado el Coordinador podremos hacer lo mismo con el otro módulo, el End Device, estableciendo en la ventana de ajustes los valores 9600 en el apartado Baudios Rate y 1234 para PAN ID. En la actualización del firmware no importa que establezcamos la 1.2.XX o la 1.3.XX, ya que el dispositivo Xbee que se encargará de trabajar con las tramas API es el coordinador. En nuestro caso hemos establecido la versión 1.2.XX que hará operar a nuestro módulo remoto como router o end device y en modo transparente.

- Modelo: XB24-B
- Modo de operación: ZNet 2.5 Router / End Device
- Versión de Firmware: 1247

Hecha la configuración inicial podemos establecer la función que tendrán nuestras entradas o salidas. En el caso que nos ocupa nuestro Xbee tendrá habilitada como entradas digitales los pines 11 (DIO4) y 17 (DIO3), correspondientes al sensor ultrasonidos HC-SR04 y a un pulsador respectivamente. La configuración se realiza en el apartado *I/O Settings* de la ventana de ajustes. D4 para el pin 11 y D3 para el pin 17. De la misma manera, establecemos el pin número 19 (DIO1) como salida digital en estado lógico bajo. Esto quiere decir que en D1 seleccionaremos la opción cuatro.

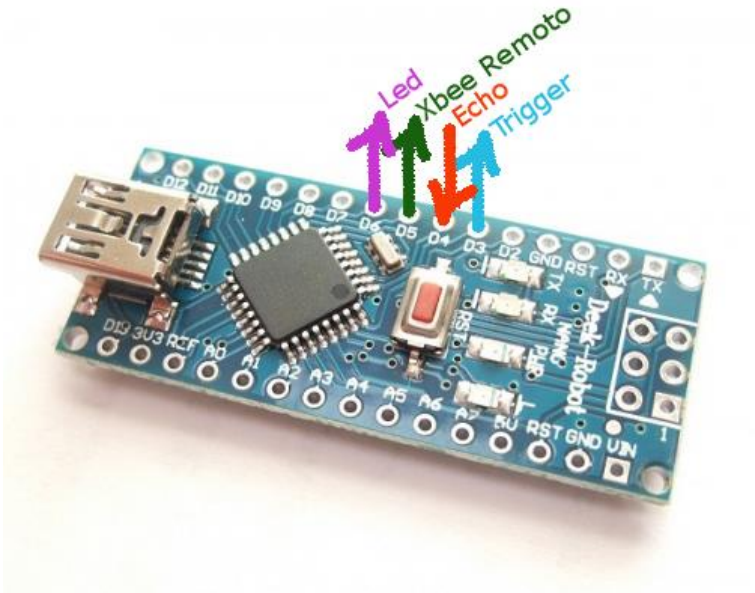


[Fig. 14] Xbee remoto: Pines utilizados

3.3. Configuración de Arduino

La configuración de nuestro dispositivo Arduino Nano consta de dos partes: las conexiones elegidas y el programa creado. Para lo primero se han elegido los pines 3, 5 y 6 como salidas digitales y el pin número 4 como entrada digital. Para lo segundo se ha utilizado el entorno de desarrollo (Arduino Software IDE) que la propia Arduino distribuye en su web: *www.arduino.cc*. Esta contiene versiones para Microsoft Windows, Mac OS y GNU/Linux. El código se ha escrito en el lenguaje de programación C y una vez cargado en el dispositivo nos permitirá controlar el sensor de ultrasonidos.

Patillaje/Pinout: Trigger (3-OUT), Echo (4-IN), Xbee (5-OUT), Led (6-OUT).



[Fig. 15] Arduino Nano v3: Conexiones elegidas

Programa:

El comienza con la declaración de las variables a utilizar. Primero establecemos las relacionadas con el procedimiento de medición del propio sensor. Estas son de tipo entero (int), permitiéndonos así almacenar valores con signo comprendidos entre -2.147.483.648 a 2.147.483.647. La variable entera LÍMITE se ha establecido en 80, ya que suele ser el ancho estándar de puertas y pasillos. Este valor es orientativo y puede ser modificado en cualquier momento, adaptándolo así a nuestras necesidades. También se han declarado variables con el nombre de cada conexión y el correspondiente pin al que pertenecen. Esto nos ayudará a ver el código de una manera más simple. [3]

```
//Declaración de variables de medida:
int distancia;
int tiempo;
int LIMITE = 80;
//Declaración de variables para los pines:
int Trigger = 3;
int Echo = 4;
int Xbee = 5;
int Led = 6;
```

En la siguiente parte del código introducimos la función *setup*, esta se ejecutará una sola vez. Estableceremos la velocidad de la conexión serial a 9600 baudios. Además, definimos la función de los pines utilizado con algunas de las variables anteriores.

```
void setup(){
//Inicialización serial a 9600 baudios:
  Serial.begin(9600);
//Funcionamiento de los pines:
  pinMode(Trigger, OUTPUT);
  pinMode(Echo, INPUT);
  pinMode(Xbee, OUTPUT);
  pinMode(Led, OUTPUT);
}
```

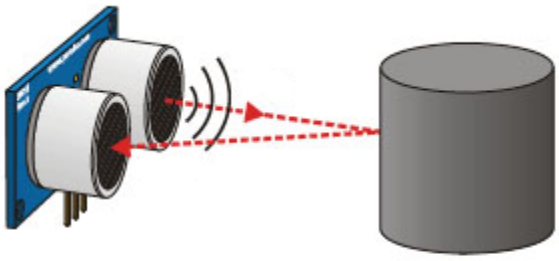
La función *loop* se ejecutará en bucle mientras la placa esté encendida. Está compuesta de dos partes: medición y evaluación de resultados. Para la primera parte construimos el disparo

del pulso sonoro poniendo en estado lógico 1 el pin *Trigger*. A continuación, gracias a la función *pulseIn* sabremos el tiempo que ha tardado la onda en regresar y poner en estado lógico alto el pin de entrada *Echo*. Este dato, junto con la velocidad a la que se desplaza el sonido (343 m/s) nos permite conocer la distancia de ida y vuelta recorrida. Para saber con precisión a qué distancia se encuentra el objeto simplemente debemos dividir entre dos el tiempo tardado y multiplicarlo por la velocidad, en cm/µs, del pulso sonoro:

```
void loop(){
  digitalWrite(Trigger, LOW);
  delayMicroseconds(5);
//Onda ultrasonido: Envío del pulso
  digitalWrite(Trigger, HIGH);
  delayMicroseconds(10);
//Mediciones:
  tiempo = pulseIn(Echo, HIGH);
  distancia = 0.017*tiempo;
```

$$Velocidad\ del\ sonido = 343\ \frac{m}{s} \cdot 100\frac{cm}{m} \cdot \frac{1}{1000000}\frac{s}{\mu s} = \mathbf{0.0343}\frac{\boldsymbol{cm}}{\boldsymbol{\mu s}}$$

$$Distancia\ (cm) = \frac{Tiempo\ (\mu s) \cdot 0.0343\frac{cm}{\mu s}}{2} = \boldsymbol{Tiempo\ (\mu s) \cdot 0.0172\ \frac{cm}{\mu s}}$$



Tiempo = 2 * (Distancia / Velocidad)
Distancia = Tiempo · Velocidad / 2

[Fig. 16] Sensor ultrasonidos: explicación gráfica

La segunda parte de la función *loop* tiene como misión evaluar los resultados obtenidos en la medición. Si esta es inferior al valor establecido en la variable LÍMITE pondrá en estado lógico alto las salidas digitales correspondientes al módulo Xbee (pin 5) y al led (pin 6). En

cambio, si la distancia medida es superior al valor de LÍMITE estos pines se mantendrán en estado lógico bajo. El programa finalizará para uno u otro caso con una espera de 1 segundo antes de volver a comenzar.

```
if (distancia < LIMITE){
     Serial.println("Distancia MENOR al límite: ");
     Serial.println(distancia);
     Serial.println(" cm");
//Órdenes a ejecutar:
     digitalWrite(Xbee, HIGH);
     digitalWrite(Led, HIGH);
     }
  if (distancia >= LIMITE){
     Serial.println("Distancia correcta: ");
     Serial.println(distancia);
     Serial.println(" cm");
//Estado inicial
     digitalWrite(Xbee, LOW);
     digitalWrite(Led, LOW);
     }delay(1000);
```

3.4.Configuración de GNU/Linux

Durante este apartado describiremos como realizar la configuración de nuestro entorno GNU/Linux. Hablaremos también de la instalación del software X-CTU y del entorno de desarrollo integrado de la placa Arduino.

Configuración de Debian GNU/Linux:

Se utilizará la versión 8.0 de Debian, con nombre en clave Jessie. Se puede descargar desde la propia página web del proyecto Debian (*www.debian.org*) y su instalación se puede hacer de forma sencilla e intuitiva con la ayuda del *asistente de instalación*.

La configuración del sistema operativo pasa por instalar una serie de paquetes con los que poder compilar el programa de control creado, trabajar cómodamente desde la plataforma Git-Hub, establecer comunicación serial con scripts de Python y facilitar la creación de paquetes Debian. Para instalarlas nos valdremos del sistema de gestión de paquetes proporcionado por

Debian, el *Advanced Packaging Tool* (Herramienta Avanzada de Empaquetado), abreviado como APT.

El primer paquete a instalar es *gcc*, la cual nos permite compilar en GNU scripts escritos en lenguaje C. La segunda biblioteca es *git-core* y corresponderá a la plataforma de desarrollo colaborativo Git-Hub. En mi caso particular me sirvió para analizar la evolución del código que me encontraba desarrollando y, además, me valió de enlace de comunicación con mi tutor de proyecto. El siguiente que se ha instalado sirve para poder realizar una conexión serial con scripts escritos en el lenguaje Python. Estos scripts han sido suministrados por el tutor a modo de sustitutos del software X-CTU en lo que a configuración de entradas y salidas se refiere. Por último instalaremos build-essentials, herramienta clave que contiene una lista informativa de los paquetes que se consideran esenciales para la creación de paquetes Debian. Esta nos servirá, entre otras cosas, para poder instalar el software de Arduino. El procedimiento es el siguiente:

```
apt-get install gcc build-essentials
apt-get install git-core
apt-get install python-serial
```

Es recomendable, antes de instalar las bibliotecas nombradas, realizar una actualización del listado de paquetes existentes en APT con el comando *update* y actualizar los mismos a la última versión existente en ese listado con el comando *upgrade*.

```
apt-get update
apt-get upgrade
```

Si existiera alguna duda sobre el nombre de un paquete siempre podemos buscarlo con la siguiente instrucción:

```
apt-cache search nombrepaquete
```

Por último, se instalará y configurará la aplicación *mailutils*, que sirve para enviar correos electrónicos desde un terminal de GNU/Linux. También será necesario instalar el paquete *SSMTP*. Lo haremos de la siguiente manera:

```
apt-get install mailutils ssmtp
```

La configuración se llevará acabo editando, con por ejemplo el editor de textos gedit, el archivo *ssmtp.conf* ubicado en la ruta */etc/ssmtp*. Para ello se modificará lo establecido a continuación: [10] [13]

```
gedit /etc/ssmtp/ssmtp.conf
```

Parámetros a editar:

- root=xbeetfg@gmail.com
- mailhub=smtp.gmail.com:587
- hostname=xbeetfg@gmail.com
- AuthUser=xbeetfg
- AuthPass=contraseña
- UseTLS=Yes
- UseSTARTTL=Yes

Configurar X-CTU de Digi International:

La versión de X-CTU para GNU/Linux viene en un fichero ejecutable con extensión ***.run***. Para instalarlos primero debemos entrar a la consola como súper-usuarios (root), situarnos en la carpeta que contiene el fichero y proporcionarle permisos de ejecución. Seguidamente procedemos a ejecutar el archivo de instalación desde el terminal con la instrucción '**./**'.

```
chmod 777 nombrefichero.run
./nombrefichero.run
```

Configurar Arduino Software IDE:

El fichero de instalación descargado desde *www.arduino.cc* tiene extensión ***.sh*** y viene comprimido en un archivo ***tar.xz***. Para instalar un fichero ejecutable como los ***.sh*** debemos proceder exactamente igual que con los ***.run***, es decir, primero utilizamos el comando *chmod* con el parámetro '*777*' para darle permisos de ejecución, lectura y escritura a cualquier usuario y luego ejecutamos el fichero con la instrucción '**./**'.

3.5.Configuración de Raspberry Pi

Esta configuración se realizará exactamente igual que con el equipo informático anterior. Se conectará el dispositivo Xbee coordinador mediante la placa Xbee USB Explorer a la Raspberry Pi B de igual manera que se hizo con el equipo que soportaba Debian GNU/Linux.

Se ha elegido la distribución Raspbian precisamente por su relación y parecido con la distro Debian GNU/Linux montada en nuestro ordenador. La descarga del sistema operativo se puede realizar desde la propia web de la Fundación Raspberry Pi, *www.raspberrypi.org*. Bajaremos la versión Jessie y grabaremos su imagen *.iso* en una tarjeta micro-SD con ayuda de software como YUMI o LiveUSB Install. Al iniciar nuestra RPi se nos mostrará un menú de

configuración inicial en el que podremos realizar modificaciones para entre otras cosas: extender la partición root y usar así todo el espacio de nuestra tarjeta SD, cambiar la contraseña, configurar nuestro teclado, zona horaria, etc. Si este menú no saliera por defecto siempre podremos acceder a él escribiendo en un terminal (en modo súper-usuario) la instrucción *raspi-config*.

La conexión entre la Raspberry Pi y el módulo Xbee Coordinador será, como se ha mencionado antes, mediante la mini-placa Xbee USB Explorer. Es decir, los interconectaremos mediante uno de los puertos USB que proporciona nuestra placa mini-PC. No obstante, también sería posible establecer esta comunicación serial mediante los pines de transmisión y recepción de ambos dispositivos, pero para esto es probable que necesitemos instalar en nuestro mini-ordenador alguna biblioteca adicional como WiringPi.

A partir de este punto podemos configurar el sistema, a través de la consola, tal y como se hizo con el Debian GNU/Linux de nuestro ordenador personal. Es decir, debemos añadir los paquetes *gcc*, *build-essentials*, *git-core* y *python-serial*.

De manera teórica, se ha indagado brevemente en cómo utilizar los pines de transmisión y recepción que nos ofrece nuestra placa para establecer la comunicación con el Xbee Coordinador. De esta manera no se ocuparía uno de los puertos USB. Para conseguirlo, un posible camino sería instalar y añadir a nuestro código la biblioteca *WiringPi* (adicionalmente también se podría incluir la biblioteca *WiringSerial*). Esta dará soporte a nuestra Raspberry Pi para que sea capaz de utilizar el puerto serie de la tarjeta (GPIO de Tx y Rx). La biblioteca WiringPi se encuentra escrita en lenguaje C y liberada bajo una licencia GNU. Su principal función es permitir la programación de periféricos que se encuentren conectados a nuestra placa mediante los pines de propósito general (GPIO). El procedimiento para instalar y utilizar esta biblioteca es el siguiente:

Instalación WiringPi:

Para la instalación debemos descargar la última versión disponible de la biblioteca a través de Git-Hub. Para hacerlo utilizamos el comando *git clone*, disponible gracias a que previamente hemos instalado el paquete *git-core*.

```
git clone git://git.drogon.net/wiringPi
```

A continuación, nos desplazamos a la carpeta con la instrucción *cd* y descargamos la

última versión disponible con el comando *pull* del paquete *git-core*.

```
cd wiringPi
git pull origin
```

Por último, instalamos WiringPi ejecutando el fichero *build*:

```
./build
```

Modo de empleo:

Para poder utilizar esta biblioteca es necesario indicar en el script donde se va a emplear y el puerto al que se debe conectar. Sustituiremos 'canal' por */dev/ttyUSB0* y 'baudios' por *9600*.

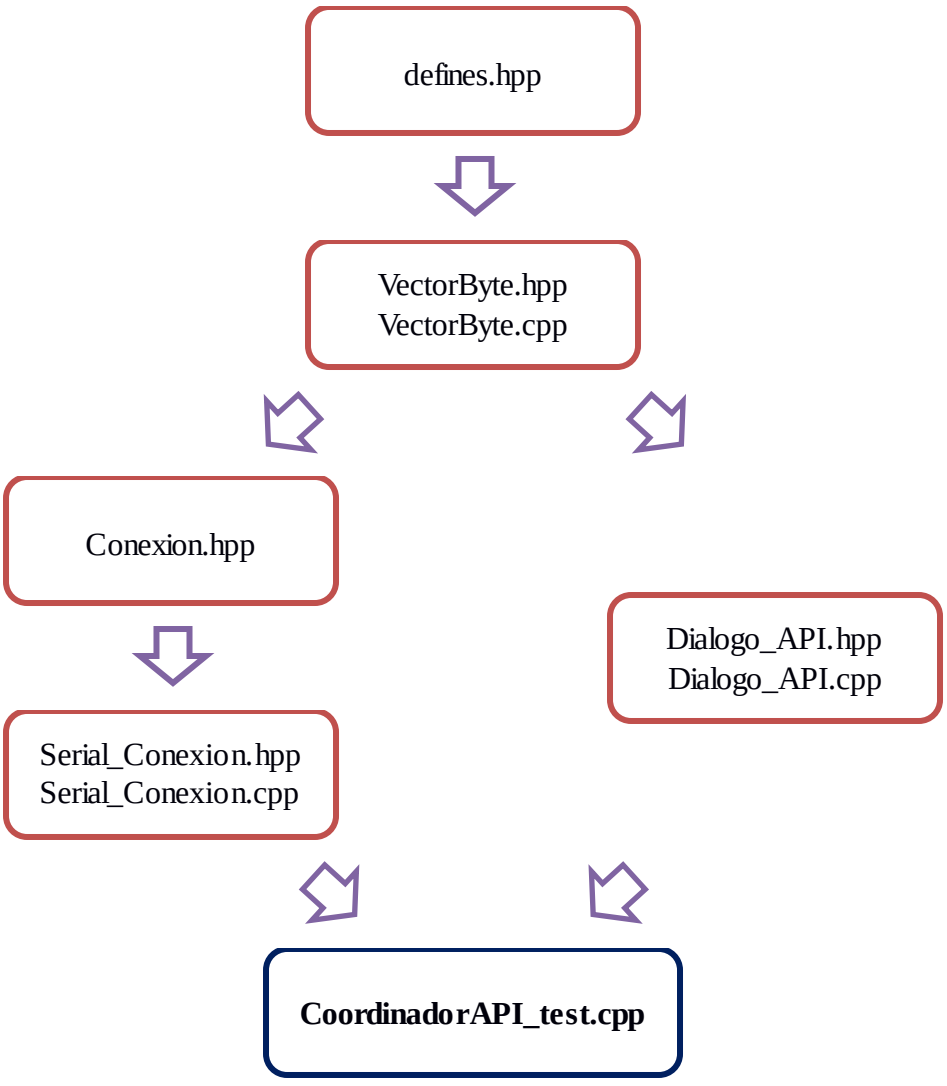
```
#include <wiringPi.h>
int wiringPiSPISeutp(int canal, int baudios)
```

Por último, debemos compilar el script añadiendo *–l wiringPi* a la instrucción: [12] [15]

```
gcc-o CoordinadorAPI_test –l wiringPi CoordinadorAPI_test
```

3.6.Programa desarrollado

Para el desarrollo de este programa se ha utilizado el lenguaje de programación C++, que nos permite la manipulación de objetos. Durante el proceso nos hemos apoyado en una serie de ficheros escritos en C++ por el tutor que nos facilitarán la comunicación serial con el módulo Xbee Coordinador y la interpretación y tratamiento de las tramas API creadas. Asimismo, cada clase de estos se codifica en dos ficheros fuente diferente: los que poseen extensión *.hpp* se encargan de declarar las distintas clases existentes y los de extensión *.cpp* de definir los métodos de estas. La estructura jerárquica del conjunto de ficheros, programa desarrollado inclusive, sería la siguiente:



[Fig. 17] Estructura jerárquica de los ficheros

El primero de los archivos, *defines.hpp*, define la variable de tipo *byte* que será utilizada por el resto de ficheros. *VectorByte* trata de manejar el conjunto de bytes que nos llegan, permitiéndonos almacenarlos y representarlos como una trama API. La comunicación

ordenador-Xbee es posible gracias a los ficheros *Conexión.hpp*, *Serial_Conexion.hpp* y *Serial_Conexion.cpp*. El primero de ellos permite establecer una conexión serial genérica gracias a la clase abstracta *Conexion*. En *Serial_Conexion* se declaran las clases y se definen los métodos necesarios para que esta conexión se produzca a través del puerto serie.

Llegados a este punto ya podemos trabajar con el Xbee Coordinador, no obstante, aún no podemos comunicarnos con el otro Xbee existente en la red. *Dialogo_API* se encargará de establecer esta comunicación, permitiéndonos así interpretar y trabajar con las tramas que el modo de comunicación API utiliza. Asimismo, cuando deseamos enviar un paquete podemos utilizar la función *enviaPaquete* de la clase *DialgoAPI* y construirlo tal y como se explicó en el apartado 2.1.5. En este sentido, la clase *DialogoAPI* nos sirven, además, para enviar comandos AT directamente al nodo remoto. Esto es posible gracias a la función *comandoATremoto*, la cual construye el mensaje y luego lo envía a través de la función anterior. De igual forma, cuando se recibe una trama API podemos interpretarla gracias a la función *recibePaquete* de la clase antes mencionada.

Comenzamos nuestro código incluyendo los ficheros anteriormente nombrados y una serie de librerías que nos harán falta para, entre otras cosas, poder mostrar mensajes por pantalla, manipular cadenas de caracteres, ejecutar comandos externos o trabajar con el tamaño y la posición de los vectores. También se han definido una serie de variables referentes al color del texto mostrado en la terminal. Estas resultan realmente útiles para clarificar lo que se muestra por pantalla.

```
/*   XBEE-COORDINADOR   */
//Declaraciones:
#include "Dialogo_API.hpp"
#include "Serial_Conexion.hpp"
#include "VectorByte.hpp"
#include "defines.hpp"

//Bibliotecas:
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>

#include <iomanip>
#include <vector>
```

```
//Definiciones:
#define BVEC(v) VectorByte(v, v + sizeof(v)/sizeof(byte) )

// Colores de texto:
#define RED    "\033[31m"
#define GREEN  "\033[32m"
#define YELLOW "\033[33m"
#define BLUE   "\033[34m"
#define BLACK  "\033[0m"
```

Proseguimos con dos funciones auxiliares que definirán el funcionamiento del zumbador (buzzer). La primera hará que el dispositivo electrónico se active durante cinco segundos y luego quede desactivado. La segunda realizara lo mismo durante un periodo más corto de tiempo. Se ha optado por construir una estructura API completa, justo como lo haría la función *comandoATremoto* de la clase *DialogoAPI* declarada en el fichero que recibe el mismo nombre. Más adelante se ha optado por enviar el comando AT directamente desde la función antes mencionada, por un lado, para simplificar y por otro lado, para mostrar las dos formas de trabajar. En mensaje API que nos ocupa incluye un identificador de trama para el envío de un comando AT remoto (0x17). Este se describe más ampliamente durante el apartado 2.1.5 de la presente memoria. El comando a enviar es *D15*, este nos sirve para poner en estado lógico alto el pin número 19 y con ello activar el zumbador. Seguidamente se esperan cinco segundos gracias a la instrucción Sleep y posteriormente se desactiva el actuador poniendo en estado lógico bajo la salida digital, es decir, enviando el comando *D14*.

```
void zumbador_largo(Dialogo_API &dapi){
  byte menZumb[] = { 0x17, 0x00, 0x00, 0x13, 0xA2, 0x00, 0x40,
0x34, 0x0E, 0x6A, 0xFF, 0xFE, 0x02, 0x44, 0x31, 0x05 };
  VectorByte OBJ = BVEC(menZumb);
  dapi.enviaPaquete(OBJ);
  sleep(5);
  byte menZumb2[] = { 0x17, 0x00, 0x00, 0x13, 0xA2, 0x00,
0x40, 0x34, 0x0E, 0x6A, 0xFF, 0xFE, 0x02, 0x44, 0x31, 0x04 };
  VectorByte OBJ2 = BVEC(menZumb2);
  dapi.enviaPaquete(OBJ2);
};

void zumbador_corto(Dialogo_API &dapi){
  byte menZumb[] = { 0x17, 0x00, 0x00, 0x13, 0xA2, 0x00, 0x40,
0x34, 0x0E, 0x6A, 0xFF, 0xFE, 0x02, 0x44, 0x31, 0x05 };
  VectorByte OBJ = BVEC(menZumb);
  dapi.enviaPaquete(OBJ);
  sleep(0.5);
  byte menZumb2[] = { 0x17, 0x00, 0x00, 0x13, 0xA2, 0x00,
0x40, 0x34, 0x0E, 0x6A, 0xFF, 0xFE, 0x02, 0x44, 0x31, 0x04 };
  VectorByte OBJ2 = BVEC(menZumb2);
```

```
  dapi.enviaPaquete(OBJ2);
};
```

Como se puede comprobar en la siguiente tabla, del mensaje enviado podemos destacar el identificador de la trama, la dirección de destino y el comando AT seleccionado. El valor del parámetro es lo que definirá la activación o no de esta salida digital. Si este valor se corresponde con un 5, el buzzer será activado. En cambio, si el parámetro se define como un 4 significará que estamos enviando la orden de desactivar el zumbador.

0x17	0x00	0x00	0x13	0xA2	0x00	0x40	0x34	0x0E	0x6A	0xFF	0xFE
ID trama API		Dirección de 64 bits								Dir. de 16 bits	

0x02	0x44	0x31	0x05
Opción del comando	Comando AT: **D1**		Valor del parámetro: **5**

[Tabla 11] Activación del actuador

Definidos los modos de operar del zumbador continuamos con la función principal del cogido, la función *main*. Esta comenzará llamando a un comando externo para que limpie cualquier texto anterior mostrado en el terminal en el que se ejecutará nuestro programa. Continuará con la declaración de variables, booleanas, enteras y una cadena de caracteres, que determinarán el modo de operación de los menús, la velocidad de la conexión y el puerto serie al que nos debemos conectar para encontrar el Xbee Coordinador. Las variables de menú se mantendrán activas (en estado *true*) hasta que seleccionemos determinadas opciones que permitan desactivarlas (poner en estado *false*) y, por consecuencia, salir del menú o cerrar el programa. A continuación, se abrirá el puerto en base a los parámetros previamente definidos en las variables *baudios* y *puerto*.

```
int main(){
  system("clear");
//Variables: ·····················································
  bool menu = true;
  bool submenu = true;
  bool variable = false;
  int opcion;
  int baudios = 9600;
  std::string puerto = "/dev/ttyUSB0";

//Conexión serial: ···············································
  Serial_Conexion *serc = new Serial_Conexion();
  std::cout << BLUE << "\tUsando puerto " << puerto << " con
velocidad " << baudios << BLACK << std::endl;
```

```
serc->open_port( puerto, baudios);
Dialogo_API dapi(serc);
```

Seguidamente se crea el menú principal del programa de control. En este podremos seleccionar dos modos de operación: funcionamiento y testeo. Introducimos un 1 para el primer modo. De esta manera el programa funcionará de manera continua hasta que se accione el pulsador. Si un movimiento es detectado se enviará un email a la cuenta *XbeeTFG@gmail.com* con el asunto y mensaje expresados en el código. Este envío se hará gracias al comando externo *mail*, que nos permite enviar correos electrónicos desde un terminal de GNU/Linux. El programa es capaz de detectar estas variaciones en las entradas y salidas del módulo Xbee remoto gracias al comando AT *IS*. Este podría haber sido construido de la misma manera que la activación del zumbador anteriormente explicada, con una estructura API que contenga un indicador de trama, una dirección de 64 bits y el comando AT entre otros frames a enviar. No ha sido así, la opción empleada, como se comentó antes, se apoya en la función *comandoATremoto* de la clase *DialogoAPI* contenida en el fichero externo de mismo nombre. Envidado el comando el programa esperará indefinidamente hasta recibir un paquete válido gracias a la sentencia *do while*, la cual creará un bucle infinito hasta que se recibe un paquete de tamaño distinto de cero. De esta manera, sorteamos el tiempo máximo que los dispositivos Xbee esperan para recibir un paquete, el llamado *timeout*.

Cuando recojamos una respuesta del comando IS, que recordemos sirve para forzar la lectura de las entradas del dispositivo, nos dispondremos a analizar el mensaje recibido. Las variaciones en las entradas se encuentran en la posición 17 del vector recibido. Esta posición varía con respecto a la especificada en la sección dedicada al identificador de trama 0x92 del apartado 2.1.5, ya que en el mensaje que nos llega se omiten algunos bytes, como por ejemplo el de cabecera (0x7E).

Para evaluar el mensaje utilizaremos el operador de desplazamiento <<, este nos servirá para crear una máscara de bits que nos permita analizar solamente el bit que deseamos, en nuestro caso (siempre contando desde 0) los situados en las posiciones 3 y 4. Así pues, para comprobar el estado de estas entradas digitales las evaluaremos por separado y con ayuda del operador AND (&), que actuará como filtro. Para detectar cuando el sensor ultrasonidos es activado se comprueban los bits de la posición 4 de la máscara y del mensaje recibido, cuando sean 1 la condición evaluada en la sentencia if será verdadera y cambiará el estado booleano de la variable *pingUp* asociada a nuestro sensor. De igual manera, si queremos saber cuándo el pulsador es accionado y si, por tanto, se cumple la condición de la segunda sentencia if haremos

lo mismo que para el sensor ultrasonidos pero esta vez con una máscara que compruebe la posición número 3. La variable asociada al pulsador, *buttonUp*, se colocará en estado *false* porque se está evaluando de manera opuesta. El pulsador funciona con lógica inversa, es decir, se coloca en estado lógico bajo solamente cuando es accionado, por lo tanto su variable asociada, *buttonUp*, se mantendrá activa (estado *True*) hasta que este sea accionado y, por consecuencia, se ponga en estado *False*.

Ejemplo para el sensor ultrasonidos:

```
if((paqRecibido->at(17)) & (1<<4)) pingUp = true;
```

Caso 1	**Pos. 4**	Pos. 3	Pos. 2	Pos. 1	Pos. 0
Sensor activado	**1**	0	0	0	0
Máscara: 1<<4	**1**	0	0	0	0
AND (&)	True	False	False	False	False

Caso 2	**Pos. 4**	Pos. 3	Pos. 2	Pos. 1	Pos. 0
Sensor desactivado	**0**	0	0	0	0
Máscara: 1<<4	**1**	0	0	0	0
AND (&)	False	False	False	False	False

[Tabla 12] Ejemplo: Análisis del estado del sensor ultrasonidos

Dependiendo de lo que hayamos recibido se procederá de una forma u de otra. Si ambos pines de entrada se encuentran desactivados el programa de control continuará ejecutándose normalmente. Si el sensor ultrasonidos se ha activado entonces se pondrá en estado *true* la variable *pingUp*, se mostrará un mensaje por pantalla indicando lo que ha sucedido, se enviará el email, se llamará a la función *zumbador_largo* para que active el buzzer y se dejará en estado *false* la variable denominada *variable*. Esta última es la que permite finalizar este modo de operación y volver al menú principal. En este caso, al encontrarse en estado *false*, dejará que el programa siga funcionando de forma continua. Por otro lado, si el pulsador fuera accionado se llamará a la función *zumbador_corto* y *variable* se pondrá en estado *true*, permitiendo así volver al programa principal.

```
//Menú principal: ·····································
  while(menu == true){
    std::cout << YELLOW << "\n\tMENU PRINCIPAL:\t\t|1|¡En
marcha!\t\t|2|Testeo\t\t|"
```

```
    << RED << "?" << YELLOW << "|Terminar";
    std::cout << BLACK << "\n\t - Elegir: ";
    std::cin >> opcion;
    switch (opcion) {
      case 1:
        std::cout << "\t - CASE 0: Funcionamiento hasta que se
presione el pulsador" << std::endl;
        while (variable == false) {
          bool pingUp = false;
          bool buttonUp = true;
          VectorByte *paqRecibido;
          sleep(1);
          //Esperando paquete:
          dapi.comandoATremoto( "IS", 0x0013A20040340E6A, -1,
true);
          do {
            paqRecibido = dapi.recibePaquete();
            std::cout << "Recibido paquete de tamaño " <<
paqRecibido->size() << std::endl;
          } while( paqRecibido->size() == 0 );
          //Analizamos el paquete recibido:
          //std::cout << BLUE << *paqRecibido << BLACK <<
std::endl;
          std::cout << GREEN << paqRecibido->at(17) << BLACK
<< std::endl;

          if((paqRecibido->at(17)) & (1<<4)) pingUp = true;
          if((paqRecibido->at(17)) & (1<<3)) buttonUp = false;
          //Ejecución en función de lo obtenido:
          if(pingUp == true) {
            std::cout << BLUE << "PING recibido" << BLACK <<
std::endl;
            zumbador_largo(dapi);
            system("echo 'Movimiento detectado' | mail -s
'Asunto: Xbee-remoto' XbeeTfg@gmail.com"); //-a para incluir
fichero.ext
            sleep(1);
            variable = false;
          }
          if(buttonUp == true) {
            std::cout << BLUE << "BOTÓN presionado:" << BLACK
<< std::endl;
            std::cout << " - Escuchamos un pitido corto." <<
BLACK << std::endl;
            zumbador_corto(dapi);
            sleep(0.5);
            std::cout << " - Volvemos al menú principal." <<
BLACK << std::endl;
            variable = true;
          }
        }//end_while_case-0:variable=true
    break;
```

La segunda opción disponible en el menú principal de nuestro programa de control es la referida al testeo del sistema montado. Podemos ponerlo a prueba de dos formas: verificando el funcionamiento del zumbador o el del pulsador. Para comprobar que nuestro buzzer funciona correctamente seleccionaremos la primera opción. Entonces deberemos escuchar primero un tono largo, esperar un segundo y luego escuchar un tono más corto. Hecho esto se volverá al submenú. Para verificar el funcionamiento del pulsador y no tener dudas de que podremos interrumpir nuestro programa en cualquier momento seleccionamos la segunda opción. Seguiremos las opciones indicadas por pantalla y verificaremos que su funcionamiento es el correcto. El proceso comprobará cuatro veces el estado del pulsador durante un periodo de cuatro segundos. En caso de querer regresar al menú principal pulsaremos cualquier tecla. Esto provocará que se active la opción default de la función switch, que pondrá en estado *false* la variable submenú y utilizará el comando externo *clear* para limpiar la pantalla.

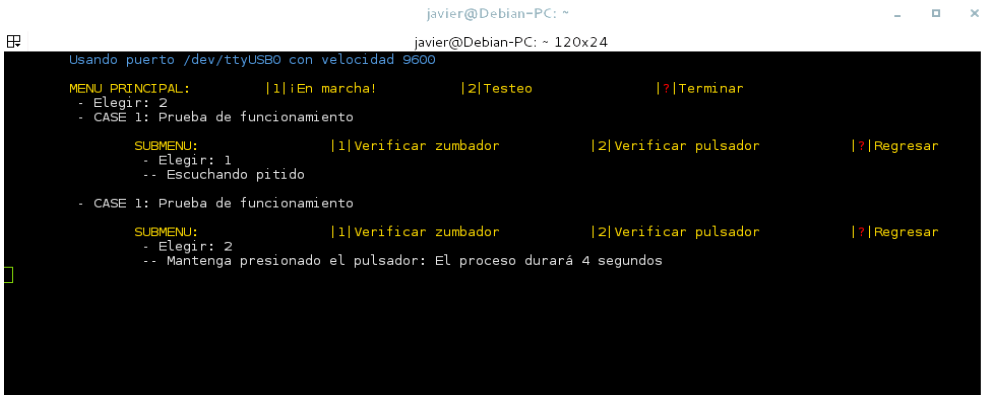
```
    case 2:
      submenu = true;
      while(submenu == true){
      int subopcion;
      std::cout << "\t - CASE 1: Prueba de funcionamiento" <<
std::endl;
      std::cout << YELLOW << "\n\t\tSUBMENU:\t\t|1|Verificar
zumbador\t\t|2|Verificar pulsador\t\t|"
      << RED << "?" << YELLOW << "|Regresar";
      std::cout << BLACK << "\n\t\t - Elegir: ";
      std::cin >> subopcion;
      //Submenú: ·····················································
      switch (subopcion) {
        case 1:
          std::cout << "\t\t -- Escuchando pitido\n" <<
std::endl;
          zumbador_largo(dapi);
          sleep(1);
          zumbador_corto(dapi);
          sleep(0.5);
        break;
        case 2:
          bool testButtonUp;
          testButtonUp = true;
          VectorByte *testPaqRecibido;
          std::cout << "\t\t -- Mantenga presionado el pulsador:
El proceso durará 4 segundos" << GREEN << std::endl;
          sleep(1);
            for(int i=0 ; i<4 ; i++){
              //Lógica del pulsador:
              dapi.comandoATremoto( "IS", 0x0013A20040340E6A, -
1, true);
              testPaqRecibido = dapi.recibePaquete();
              sleep(1);
```

```
                std::cout << BLACK << "\t\t -- " << i+1 << "
segundos" << GREEN << std::endl;
                std::cout << RED << "\t\t -- Paquete recibido: "
<< testPaqRecibido->at(17) << GREEN << std::endl;
              }
            if ( (testPaqRecibido->at(17)) & (1<<3) ) testButtonUp
= false;
            if(testButtonUp == true) std::cout << BLUE << "\n\t\t
-- Funcionamiento correcto\n" << BLACK << std::endl;
             else if (testPaqRecibido->size() < 17 || testButtonUp
== true) std::cout << RED << "\t\t -- No detectado\n" << BLACK
<< std::endl;
          break;

          default:
            submenu = false;
            system("clear");
          }//end_switch-sub
        }//end_while-sub
      break;//end_case-2
```

Por último, finalizamos nuestro programa de control con la opción default de nuestro switch principal. Esta, al presionarse cualquier tecla, se ocupará de cerrar el programa estableciendo en estado *false* la variable menú y mostrando un cajetín (con caracteres ASCII) con el mensaje "PROGRAMA CERRADO" por pantalla.

```
      default:
        std::cout << GREEN << "\n\t\t  ╔═.●.═══════╗" << BLACK <<
std::endl;
        std::cout << RED << "\t\tPROGRAMA CERRADO" << BLACK <<
std::endl;
        std::cout << GREEN << "\t\t  ╚═══════.●.═╝\n" << BLACK <<
std::endl;
        menu = false;
      }//end_switch-principal
    }//end_while-principal
}//end_main
```



[Fig. 18]Programa de control desarrollado

4. PRESUPUESTO

Artículo	Web de compra	Cantidad	Precio unitario	**TOTAL**
Módulo Xbee Serie 2	sparkfun.com	2	28.82 €	**57.64 €**
Xbee USB Explorer	sparkfun.com	1	24.95 €	**24.95 €**
Breakout board*	sparkfun.com	1	2.95 €	**2.95 €**
Raspberry Pi Modelo B	farnell.com	1	26.39 €	**26.39 €**
Arduino Nano v3	dx.com	1	3.50 €	**3.50 €**
Módulo alimentación MB102	dx.com	1	2.26 €	**2.26 €**
Sensor Ultrasonidos HC-SR04	dx.com	1	1.82 €	**1.82 €**
Zumbador/Buzzer	dx.com	1	4.05 €	**4.05 €**
				123.56 €

[Tabla 13] Presupuesto material

***Brekout board:** Adaptador de espesor y separación de pines para Xbee.

5. CONCLUSIÓN

5.1.Conclusiones

Se ha estudiado como configurar una red Zigbee básica a través de los dispositivos Xbee. Esta se ha complementado con una serie de sensores y un actuador que han hecho realidad el proyecto de introducción a la domótica planteado. Igualmente, se han configurado distintos sistemas como Debian GNU/Linux o Raspbian, se ha programado una placa Arduino, se ha desarrollado un software de control en C++ y se ha seleccionado el sensor y el actuador más apropiado para el objetivo planteado.

5.2.Conclusions

I have studied about how to configure a basic Zigbee network using Xbee devices. This network has been complemented with some sensors and actuator that have made possible this proyect. In addition I have configure differents systems as Debian GNU/Linux or Raspbian, I have proggramed on an Arduino device, I have developed a control software on C++ language, and I have selected the most appropriated sensor and actuator for this circuit.

5.3.Líneas abiertas

En este trabajo fin de grado se han abordado parte de las posibilidades que ofrecen los dispositivos Xbee, como el control de las diferentes entradas y salidas digitales en un nodo remoto. Una continuación de este proyecto podría indagar en la configuración de las entradas analógicas con sensores del mismo tipo e investigar las posibilidades que ofrece la modulación por ancho de pulso de las salidas PWM. Otra de las líneas abiertas existentes es la configuración de redes Zigbee más amplias y complejas que la estudiada, incluyendo así nuevos sensores y actuadores, además de nuevos módulos Xbee. Asimismo, se podría estudiar el complementar el sistema creado con aplicaciones para dispositivos móviles que nos permitan controlar nuestra red personal desde el exterior. Por último, otra de las líneas que ha quedado abierta es la referida a la comunicación mediante los pines UART. Estas conexiones de transmisión y recepción existen tanto en los dispositivos Xbee como en las placas Arduino y Raspberry Pi. Un buen camino de investigación sería el interconectar, y por tanto comunicar, estos dispositivos mediante esos pines para disminuir así el cableado existente en, por ejemplo, los circuitos remoto

6. BIBLIOGRAFÍA

[1] A1117, URL: www.alldatasheet.com

[2] Digi International. Zigbee Wireless Standard, URL: http://www.digi.com/resources/standards-and-technologies/rfmodems/zigbee-wireless-standard

[3] Educachip. HC-SR04 Arduino, URL: http://www.educachip.com/hc-sr04-arduino-tutorial/

[4] Esquemático MB-102, URL: http://www.nextiafenix.com/wp-content/uploads/2014/06/MB102-esquematico.pdf

[5] Esquemático Xbee-Explorer, URL: https://cdn.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Explorer-v21b.pdf

[6] FT231X, URL: http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT231X.pdf

[7] Lanfranconi Peña, Gustavo. Diseño de varios sensores y actuadores domóticos que utilicen comunicación inalámbrica ZigBee, URL: http://riull.ull.es/xmlui/handle/915/668

[8] LM317, URL: www.alldatasheet.com

[9] MIC5219, URL: www.alldatasheet.com

[10] Nireleku. Enviar correos desde la terminal, URL: https://www.nireleku.com/2013/01/enviar-correos-desde-la-terminal-con-mail-en-ubuntu-server-12-04-lts/

[11] Oyarce Andrés. Guía del usuario Xbee Serie 1, URL: http://docplayer.es/

[12] Rpiplus. Librería WiringPi, URL: http://rpiplus.blogspot.com.es/2013/06/libreria-wiring-pi.html

[13] Sobrebits. Enviar correos desde la línea de comandos con sSMTP y Gmail, URL: http://sobrebits.com/enviar-correos-desde-la-linea-de-comandos-con-ssmtp-y-gmail/

[14] Sparkfunk. XBee® ZNet 2.5/XBee-PRO® ZNet 2.5 OEM RF Modules, URL: https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-2.5-Manual.pdf

[15] WiringPi. GPIO Interface library for the Raspberry Pi, URL: http://wiringpi.com/

[16] Wikipedia. Zigbee, URL: https://en.wikipedia.org/wiki/ZigBee